Data Analytics for Non-Life Insurance Pricing

– Lecture Notes –

Mario V. Wüthrich
RiskLab Switzerland
Department of Mathematics
ETH Zurich

Christoph Buser
AXA
versicherungen

Version October 27, 2021

Preface and Terms of Use

Lecture notes. These notes are the basis of our lecture on Data Analytics for Non-Life Insurance Pricing at ETH Zurich. They aim at giving insight and education to practitioners, academics and students in actuarial science on how to apply machine learning methods for statistical learning. There is some overlap in these notes with our book Statistical Foundations of Actuarial Learning and its Applications which treats the topic of deep learning in much more depth, see [141]

Prerequisites. The prerequisites for these lecture notes are a solid education in mathematics, in particular, in probability theory and statistics. Moreover, knowledge in statistical software such as R is required.

Terms of Use. These lecture notes are an ongoing project which is continuously revised and updated. Of course, there may be errors in the notes and there is always room for improvement. Therefore, we appreciate any comment and/or correction that readers may have. However, we would like you to respect the following rules:

- These notes are provided solely for educational, personal and non-commercial use. Any commercial use or reproduction is forbidden.
- All rights remain with the authors. They may update the manuscript or withdraw the manuscript at any time. There is no right of availability of any (old) version of these notes. The authors may also change these terms of use at any time.
- The authors disclaims all warranties, including but not limited to the use or the contents of these notes. On using these notes, you fully agree to this.
- Citation: please use the SSRN URL `https://ssrn.com/abstract=2870308`
- All included figures were produced by the authors with the open source software R.

Versions of the 1st edition of these notes (before 2019):
November 15, 2016; January 27, 2017; March 28, 2017; October 25, 2017; June 11, 2018

Versions of the 2nd edition of these notes (after 2019):
February 5, 2019; June 4, 2019; September 10, 2020

Acknowledgment

Firstly, we thank RiskLab and the Department of Mathematics of ETH Zurich, who have always strongly supported this project. A special thank you goes to Peter Reinhard (AXA Insurance, Winterthur) who has initiated this project by his very supportive and forward-looking anticipation.

Secondly, we kindly thank Patrick Zöchbauer who has written his MSc ETH Mathematics thesis under our supervision at AXA-Winterthur on "Data Science in Non-Life Insurance Pricing: Predicting Claims Frequencies using Tree-Based Models". This MSc thesis has been a great stimulus for these lecture notes and it has also supported us to get more easily into this topic. This thesis is awarded the Walter Saxer Insurance Price 2016.

Next we thank the Institute of Statistical Mathematics (ISM), Tokyo, and, in particular, Prof. Tomoko Matsui (ISM) and Prof. Gareth Peters (University College London, UCL, and Heriot-Watt University, Edinburgh) for their support. Part of these notes were written while MVW was visiting ISM.

We kindly thank Philippe Deprez, John Ery and Andrea Gabrielli who have been carefully reading preliminary versions of these notes. This has helped us to substantially improve the outline.

Finally, we thank many colleagues and students for very fruitful discussions, providing data and calculating examples. We mention in particular: Michel Baes, Peter Blum, Hans Bühlmann, Peter Bühlmann, Patrick Cheridito, Philippe Deprez, Paul Embrechts, Andrea Ferrario, Andrea Gabrielli, Guojun Gan, Guangyuan Gao, Donatien Hainaut, Christian Lorentzen, Nicolai Meinshausen, Michael Merz, Alexander Noll, Gareth Peters, Ronald Richman, Ulrich Riegel, Robert Salzmann, Jürg Schelldorfer, Pavel Shevchenko, Olivier Steiger, Qikun Xiang, Xian Xu.

Zurich, October 27, 2021 Mario V. Wüthrich & Christoph Buser

Contents

Chapter 1

Introduction to Non-Life Insurance Pricing

1.1 Introduction

The non-life insurance section ASTIN (**A**ctuarial **ST**udies **I**n **N**on-life insurance) of the International Actuarial Association (IAA) has launched a working party on big data and data analytics in 2014. The resulting document [4] states in its introduction:

> "The internet has changed and continues to change the way we engage with the tangible world. The resulting change in communication platforms, commercial trade and social interactions make the world smaller and the data bigger. Whilst the fundamentals of analyzing data have not changed, our approach to collating and understanding data, creating accessible and useful information, developing skill sets and ultimately transforming huge and ever-growing repositories of data into actionable insights for our employers, shareholders and our communities more generally has entered a new paradigm.
> As a corollary, this has made the fundamentals of data processing, modeling techniques and aligning business structures accordingly more important - no longer can existing approaches suffice - we now face the pressures of a faster moving world, blurring business lines which make customer-centricity surpass a siloed product/service focus and challenges to the actuary being central to predictive modeling.
> We too need to evolve, working intelligently with a wide cross-function of skill sets such as data experts, data scientists, economists, statisticians, mathematicians, computer scientists and, so as to improve the transformation of data to information to customer insights, behavioral experts. This is a necessary evolution for actuaries and the profession to remain relevant in a high-tech business world."

The goal of these lecture notes is to face this paradigm. We aim at giving a broad toolbox to the actuarial profession so that they can cope with these challenges, and so that they remain highly competitive in their field of competence. In these lecture notes we start from the classical actuarial world of generalized linear models, generalized additive

models and credibility theory. These models form the basis of the deeper statistical understanding. We then present several machine learning techniques such as neural networks, regression trees, bagging techniques, random forests and boosting machines. One can view these machine learning techniques as non-parametric and semi-parametric statistics approaches, with the recurrent goal of optimizing a given objective function to receive optimal predictive power. In a common understanding we would like to see these machine learning methods as an extension of classical actuarial models, and we are going to illustrate how classical actuarial methods can be embedded into these non-parametric machine learning approaches, benefiting from both the actuarial world and the machine learning world. These lecture notes have also served as a preliminary version to our book [141] which treats the topic of statistical modeling and deep learning in much more depth, and the reader will notice that there is some (unavoidable) overlap between these lecture notes and our book [141].
A second family of methods that we are going to meet in these lecture notes are so-called unsupervised learning methods (clustering methods). These methods aim at finding common structure in data to cluster these data. We provide an example of unsupervised learning by analyzing telematics car driving data which poses the challenge of selecting feature information from high frequency data. For a broader consideration of unsupervised learning methods we refer to our tutorial [107].

We close this short introductory section by briefly reviewing major developments identified in the China InsurTech Development White Paper [27]. The notion of Insurance Technology (InsurTech) is closely related to Financial Technology (FinTech), and it is a commonly used term for technology and innovation in the insurance industry. Among other things it compromises the following key points:

- artificial intelligence, machine learning and statistical learning, which may learn and accumulate useful knowledge through data;
- big data analytics, which deals with fact that data may be massive;
- cloud computing, which may be the art of performing real-time operations;
- block chain technology, that may be useful for a more efficient and anonymous exchange of data;
- internet of things, which involves the integration and interaction of physical devices (e.g. wearables, sensors) through computer systems to reduce and manage risk.

The actuarial profession has started several initiatives in data analytics and machine learning to cope with these challenges. We mention the working party "Data Science" of the Swiss Association of Actuaries (SAV) that aims at building a toolbox for the actuarial profession:

`https://www.actuarialdatascience.org/`

The SAV working party has prepared several tutorials on actuarial developments in machine learning. These tutorials are supported by computer code that can be downloaded from GitHub.[1] In the present lecture notes we work with synthetic data for which we exactly know the true data generating mechanism. This has the advantage that we can explicitly back-test the quality of all our models presented. In typical real world applications this is not the case and one needs to fully rely on estimated models. This is demonstrated in the tutorials of the SAV working party which describe several statistical modeling techniques on real insurance data, see [43, 89, 101, 119, 120]. We highly recommend readers to perform similar analysis on their own real data, because one crucial pillar of statistical modeling is to derive the right intuition and understanding for the data and the models used.

1.2 The compound Poisson model

In this section we introduce the basic statistical approach for modeling non-life insurance claims. This approach splits the total claim amount into a compound sum which accounts for the number of claims and determines the individual claim sizes.

1.2.1 Model assumptions and first properties

The classical actuarial approach for non-life insurance portfolio modeling uses a compound random variable with N describing the number of claims that occur within a fixed time period and $Z_1, \ldots, Z_N$ describing the individual claim sizes. The total claim amount in that fixed time period is then given by

$$S = Z_1 + \ldots + Z_N = \sum_{k=1}^{N} Z_k.$$

The main task in non-life insurance modeling is to understand the structure of such total claim amount models. The standard approach uses a compound distribution for S.

Throughout, we assume to work on a sufficiently rich probability space $(\Omega, \mathcal{F}, \mathbb{P})$.

Model Assumptions 1.1 (compound distribution)**.** *The total claim amount S is given by the following compound random variable on $(\Omega, \mathcal{F}, \mathbb{P})$*

$$S = Z_1 + \ldots + Z_N = \sum_{k=1}^{N} Z_k,$$

with the three standard assumptions:

(1) N is a discrete random variable which takes values in $\mathbb{N}_0$;

(2) $Z_1, Z_2, \ldots$ are independent and identically distributed (i.i.d.);

(3) N and $(Z_1, Z_2, \ldots)$ are independent.

[1] `https://github.com/JSchelldorfer/ActuarialDataScience`

Remarks.

- If S satisfies the three standard assumptions *(1)-(3)* of Model Assumptions 1.1 we say that S has a *compound distribution*. N is called *claims count* and Z_k, $k \geq 1$, are the individual *claim sizes* or *claim severities*.
- The compound distribution has been studied extensively in the actuarial literature. The aim here is to give more structure to the problem so that it can be used for answering complex pricing questions on heterogeneous insurance portfolios.

In the sequel we assume that the claims count random variable N can be modeled by a Poisson distribution. We therefore assume that there exist an *expected (claims) frequency* $\lambda > 0$ and a volume $v > 0$.

We say that N has a Poisson distribution, write $N \sim \text{Poi}(\lambda v)$, if

$$\mathbb{P}[N = k] = e^{-\lambda v} \frac{(\lambda v)^k}{k!} \qquad \text{for all } k \in \mathbb{N}_0.$$

The volume $v > 0$ often measures the time exposure in yearly units. Therefore, throughout these notes, the volume v is called *years at risk*. If a risk is insured part of the year, say, 3 months we set $v = 1/4$.

For a random variable Z we denote its coefficient of variation by $\text{Vco}(Z) = \text{Var}(Z)^{1/2}/\mathbb{E}[Z]$ (subject to existence). We have the following lemma for the Poisson distribution.

Lemma 1.2. *Assume $N \sim \text{Poi}(\lambda v)$ for fixed $\lambda, v > 0$. Then*

$$\mathbb{E}[N] = \lambda v = \text{Var}(N) \qquad \textit{and} \qquad \text{Vco}(N) = \frac{\text{Var}(N)^{1/2}}{\mathbb{E}[N]} = \sqrt{1/\lambda v} \;\to\; 0 \textit{ as } v \to \infty.$$

Proof. See Proposition 2.8 in Wüthrich [135]. □

Lemma 1.3. *Assume that N_i, $i = 1, \ldots, n$, are independent and Poisson distributed with means $\lambda_i v_i > 0$. We have*

$$N = \sum_{i=1}^n N_i \;\sim\; \text{Poi}\left(\sum_{i=1}^n \lambda_i v_i\right).$$

Proof. This easily follows by considering the corresponding moment generating functions and using the independence assumption, for details see Wüthrich [135], Chapter 2. □

Definition 1.4 (compound Poisson model)**.** *The total claim amount S has a compound Poisson distribution, write*

$$S \sim \text{CompPoi}(\lambda v, G),$$

if S has a compound distribution with $N \sim \text{Poi}(\lambda v)$ for given $\lambda, v > 0$ and individual claim size distribution $Z_1 \sim G$.

Proposition 1.5. *Assume $S \sim \text{CompPoi}(\lambda v, G)$. We have, whenever they exist,*

$$\mathbb{E}[S] = \lambda v\ \mathbb{E}[Z_1], \qquad \text{Var}(S) = \lambda v\ \mathbb{E}[Z_1^2], \qquad \text{Vco}(S) = \sqrt{\frac{1}{\lambda v}}\ \sqrt{\text{Vco}(Z_1)^2 + 1}.$$

Proof. See Proposition 2.11 in Wüthrich [135]. □

Remarks.

- If S has a compound Poisson distribution with fixed expected frequency $\lambda > 0$ and fixed claim size distribution G having finite second moment, then the coefficient of variation converges to zero at speed $v^{-1/2}$ as the years at risk v increase to infinity. In industry, this property is often called diversification property.

- The compound Poisson distribution has the so-called *aggregation property* and the *disjoint decomposition property*. These are two extremely beautiful and useful properties which explain part of the popularity of the compound Poisson model, we refer to Theorems 2.12 and 2.14 in Wüthrich [135] for more details. The aggregation property for the Poisson distribution has already been stated in Lemma 1.3. It tells us that we can aggregate independent Poisson distributed random variables and we stay within the family of Poisson distributions.

- The years at risk $v > 0$ may have different interpretation in the compound Poisson context: either (i) we may consider a single risk which is insured over v accounting years, or (ii) we have a portfolio of independent compound Poisson risks and then v measures the volume of the aggregated portfolio (in years at risk). The latter uses the aggregation property which says that also the aggregated portfolio has a compound Poisson distribution (with volume weighted expected frequency), see Theorem 2.12 in Wüthrich [135].

One crucial property of compound distributions is that we have the following decomposition of their expected values

$$\mathbb{E}[S] = \mathbb{E}[N]\ \mathbb{E}[Z_1] = \lambda v\ \mathbb{E}[Z_1],$$

where for the second identity we have used the Poisson model assumption, see Proposition 1.5, and where we assume $S \in L^1(\Omega, \mathcal{F}, \mathbb{P})$. This implies that for the modeling of the pure risk premium $\mathbb{E}[S]$ we can treat the (expected) number of claims $\mathbb{E}[N]$ and the (expected) individual claim sizes $\mathbb{E}[Z_1]$ separately in a compound model. We make the following restriction here:

> In the present notes, for simplicity, we mainly focus on the modeling of the claims count N, and we only give broad indication about the modeling of Z_k. We do this to not overload these notes.

In many situations it is more appropriate to consider volume scaled quantities. For the claims count N we may consider the *claims frequency* defined by

$$Y = \frac{N}{v}. \tag{1.1}$$

In the Poisson model, the claims frequency Y has the following two properties

$$\mathbb{E}[Y] = \lambda \qquad \text{and} \qquad \mathrm{Var}(Y) = \lambda/v.$$

From this we see that confidence bounds for frequencies based on standard deviations $\mathrm{Var}(Y)^{1/2} = \sqrt{\lambda/v}$ get more narrow with increasing years at risk $v > 0$. This is the reason why the equal balance (diversification) concept on a portfolio level works. This is going to be crucial in the subsequent chapters where we aim at detecting structural differences between different risks in terms of expected frequencies.

Anticipatory, one often rewrites (1.1) as follows

$$Y = \lambda + \varepsilon,$$

where ε is centered with variance λ/v. In this form, λ describes the *structural behavior* of Y and ε is understood as the *noise term* that describes the random deviation/fluctuation around the structural term when running the experiment. In all what follows, we try to separate the structural behavior from the random noise term, so that we are able to quantify the price level (pure risk premium) of individual insurance policies.

1.2.2 Maximum likelihood estimation: homogeneous case

Assume that N has a Poisson distribution with volume $v > 0$ and expected frequency $\lambda > 0$, that is, $N \sim \mathrm{Poi}(\lambda v)$. For predicting this random variable N, we would typically like to use its expected value $\mathbb{E}[N] = \lambda v$ as predictor because this minimizes the mean square error of prediction (MSEP); we highlight this in more detail in Section 1.3, below. However, this predictor is only useful if the parameters are known. Typically, we know the volume $v > 0$ but, in general, we do not assume knowledge of the true expected frequency λ. Hence, we need to estimate this latter parameter.

This estimation task is solved by assuming that one has a family of independent random variables $N_1, \ldots, N_n$ with $N_i \sim \mathrm{Poi}(\lambda v_i)$ for all $i = 1, \ldots, n$. Note that for the time being we assume a *homogeneous portfolio* in the sense that all random variables N_i are assumed to have the same expected frequency $\lambda > 0$; this is going to be relaxed in the subsequent chapters. Based on this model assumption one aims at estimating the common frequency parameter λ from given observations $\boldsymbol{N} = (N_1, \ldots, N_n)'$. The joint log-likelihood function of these observations is given by

$$\lambda \;\mapsto\; \ell_{\boldsymbol{N}}(\lambda) = \sum_{i=1}^{n} -\lambda v_i + N_i \log(\lambda v_i) - \log(N_i!).$$

The parameter estimation problem is then commonly solved by calculating the maximum likelihood estimator (MLE), which is the parameter value λ that maximizes the log-likelihood function, i.e., from the class of homogeneous Poisson models the one is selected

that assigns the highest probabibility to the actual observation $\boldsymbol{N}$. Thus, the MLE $\widehat{\lambda}$ for λ is found by the solution of

$$\frac{\partial}{\partial \lambda} \ell_{\boldsymbol{N}}(\lambda) = 0. \tag{1.2}$$

This solution is given by

$$\widehat{\lambda} = \frac{\sum_{i=1}^n N_i}{\sum_{i=1}^n v_i} \geq 0. \tag{1.3}$$

Note that $\frac{\partial^2}{\partial \lambda^2} \ell_{\boldsymbol{N}}(\lambda) < 0$ for any $\lambda > 0$.[2] The MLE is unbiased for λ, i.e. $\mathbb{E}[\widehat{\lambda}] = \lambda$, and its variance is given by

$$\mathrm{Var}(\widehat{\lambda}) = \frac{\lambda}{\sum_{i=1}^n v_i},$$

which converges to zero as the denominator goes to infinity. This allows us to quantify the parameter estimation uncertainty by the corresponding standard deviation. This is going to be important in the analysis of heterogeneous portfolios, below.

1.2.3 Poisson deviance loss

Instead of maximizing the log-likelihood function we could also try to minimize an appropriate objective function. The canonical objective function under our model assumptions is the *Poisson deviance loss*. We define the maximal log-likelihood (which is received from the saturated model)

$$\ell_{\boldsymbol{N}}(\boldsymbol{N}) = \sum_{i=1}^n -N_i + N_i \log N_i - \log(N_i!). \tag{1.4}$$

The saturated model is obtained by letting each observation N_i have its own parameter $\lambda_i \stackrel{\text{def.}}{=} \mathbb{E}[N_i]/v_i$. These individual parameters are estimated by their corresponding individual MLEs $\widehat{\lambda}_i = N_i/v_i$, that is, each policy i receives its own individual MLE parameter. We set the i-th term on the right-hand side of (1.4) equal to zero if $N_i = 0$ (we use this terminology throughout these notes).

The *(scaled) Poisson deviance loss* for expected frequency $\lambda > 0$ is defined by

$$\begin{aligned} D^*(\boldsymbol{N}, \lambda) &= 2\left(\ell_{\boldsymbol{N}}(\boldsymbol{N}) - \ell_{\boldsymbol{N}}(\lambda)\right) \\ &= 2 \sum_{i=1}^n -N_i + N_i \log N_i + \lambda v_i - N_i \log\left(\lambda v_i\right) \\ &= \sum_{i=1}^n 2\, N_i \left[\frac{\lambda v_i}{N_i} - 1 - \log\left(\frac{\lambda v_i}{N_i}\right)\right] \;\geq\; 0, \end{aligned} \tag{1.5}$$

where the i-th term in (1.5) is set equal to $2\lambda v_i$ for $N_i = 0$.

Remarks.

- Each term under the summation in (1.5) is non-negative because the saturated model maximizes each of these terms individually (by choosing the individual MLEs

[2] If $\widehat{\lambda} = \sum_i N_i / \sum_i v_i = 0$, which happens with positive probability in the Poisson model, we get a degenerate model.

$\widehat{\lambda}_i = N_i/v_i$). Therefore, $D^*(\boldsymbol{N}, \lambda) \geq 0$ for any $\lambda > 0$. Note that we even receive non-negativity in (1.5) if we choose arbitrary policy-dependent expected frequencies $\lambda_i > 0$ instead of the homogeneous expected frequency parameter λ.[3]

- Maximizing the log-likelihood function $\ell_{\boldsymbol{N}}(\lambda)$ in λ is equivalent to minimizing the deviance loss function $D^*(\boldsymbol{N}, \lambda)$ in λ.
- The deviance loss can be generalized to the exponential dispersion family which contains many interesting distributions such as the Gaussian, the Poisson and the gamma models. We refer to Section 4.1.2 in [141].

We close this section by analyzing the expected deviance loss

$$\mathbb{E}[D^*(N, \lambda)] = 2\mathbb{E}\left[\lambda v - N - N \log(\lambda v/N)\right] = 2\mathbb{E}\left[N \log(N/\lambda v)\right],$$

of a Poisson distributed random variable N with expected value $\mathbb{E}[N] = \lambda v > 0$.

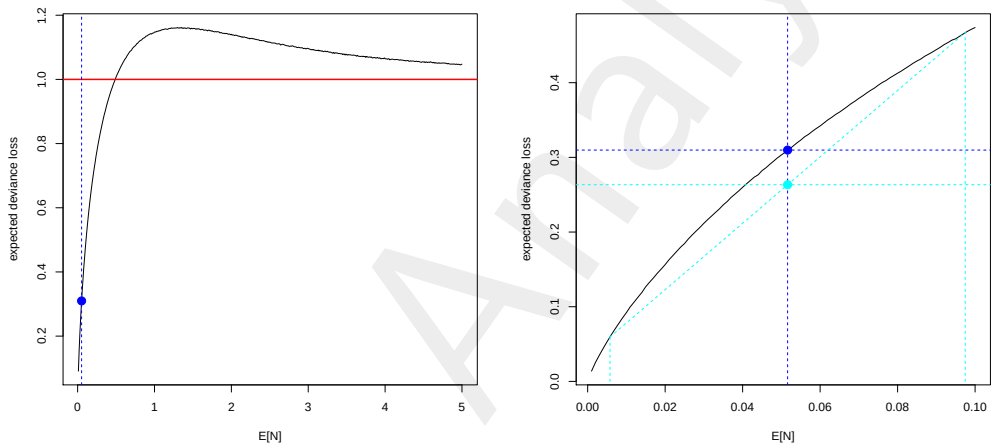


Figure 1.1: Expected deviance loss $\mathbb{E}[D^*(N, \lambda)] = 2\mathbb{E}[N \log(N/\lambda v)]$ of a claims count $N \sim \text{Poi}(\lambda v)$ as a function of its expected value $\mathbb{E}[N] = \lambda v$ on two different scales; these plots are obtained by Monte Carlo simulation, the blue dot shows mean $\mathbb{E}[N] = 5.16\%$, the vertical cyan dotted lines show means $\mathbb{E}[N] = 5.16\% \pm 4.58\%$.

Figure 1.1 illustrates this expected deviance loss on two different scales for the expected number of claims $\mathbb{E}[N] = \lambda v$. The example which we are going to study in these notes has an average expected number of claims of $\lambda v = 5.16\%$. This gives an expected deviance loss of $30.9775 \cdot 10^{-2}$ (blue dot in Figure 1.1). This is going to be important for the understanding of the remainder of these notes. Note that this value is bigger than $27.7278 \cdot 10^{-2}$ given in (A.5) in the appendix. This difference is explained by the fact that the latter value has been obtained on a heterogeneous portfolio. In this heterogeneous portfolio the individual policies have a standard deviation in expected numbers of claims of 4.58%, see Figure A.1 in the appendix. The cyan vertical dotted lines in Figure 1.1

[3] Policy dependent expected frequency parameters λ_i are going to be crucial in the subsequent chapters because the general goal of these notes is to price heterogeneous insurance portfolios.

(rhs) provide two policies with expected numbers of claims of $5.16\% \pm 4.58\%$. If we average their expected deviance losses we receive $26.3367 \cdot 10^{-2}$ which corresponds to the cyan dot in Figure 1.1 (rhs).

1.3 Prediction uncertainty

In this section we quantify prediction uncertainty. This is mainly done in terms of the Poisson model and the Poisson deviance loss. For the general case within the exponential dispersion family and more insight we refer to Chapter 4 of [141], in particular, we mention that deviance losses are strictly consistent scoring functions for mean estimation which is important in forecast evaluation, see Gneiting [55].

1.3.1 Generalization loss

Assume we have n observations, called *cases*, given by

$$\mathcal{D} = \{(N_1, v_1), \ldots, (N_n, v_n)\}. \tag{1.6}$$

$\mathcal{D}$ stands for *data*. Typically, we do *not* know the true data generating mechanism of $\mathcal{D}$. Therefore, we make a *model assumption*: assume that all n cases (N_i, v_i) are independent and that N_i are Poisson distributed with expected frequency $\lambda > 0$. We infer the expected frequency parameter λ from this data $\mathcal{D}$. We denote the resulting estimator by $\widehat{\lambda}$, for the moment this can by any (sensible) $\mathcal{D}$-dependent estimator for λ. We would like to analyze how well this estimator performs on cases which have not been seen during the estimation procedure of λ. In machine learning this is called generalization of the estimated model to unseen data, and the resulting error is called *generalization error*, *out-of-sample error* or *prediction error*.

To analyze this generalization error we choose a new random variable $N \sim \text{Poi}(\lambda v)$, which is independent from the data $\mathcal{D}$ and which has the same expected frequency λ. The frequency $Y = N/v$ of this new random variable is predicted by

$$\widehat{Y} = \widehat{\mathbb{E}}[Y] = \widehat{\lambda}. \tag{1.7}$$

Note that we deliberately write $\widehat{\mathbb{E}}$ because it is estimated from the data $\mathcal{D}$.

Proposition 1.6 (MSEP generalization loss)**.** *Assume that all cases in $\mathcal{D}$ are independent and Poisson distributed having the same expected frequency $\lambda > 0$. Moreover, let case (N, v) be independent of $\mathcal{D}$ and Poisson distributed with the same expected frequency λ. We predict $Y = N/v$ by $\widehat{Y} = \widehat{\lambda}$, where the $\mathcal{D}$-based estimator $\widehat{\lambda}$ is assumed to be square integrable. This prediction has MSEP*

$$\mathbb{E}\left[\left(Y - \widehat{Y}\right)^2\right] = \left(\mathbb{E}[Y] - \mathbb{E}\left[\widehat{Y}\right]\right)^2 + \text{Var}(\widehat{Y}) + \text{Var}(Y).$$

Proof of Proposition 1.6. The first step is done to bring the terms into the same order as in the statement of the proposition, in the second last step we use independence between $\mathcal{D}$ and (N, v),

$$\begin{aligned}
\mathbb{E}\left[\left(Y-\widehat{Y}\right)^2\right] &= \mathbb{E}\left[\left(\widehat{Y}-Y\right)^2\right] = \mathbb{E}\left[\left(\widehat{Y}-\mathbb{E}[Y]+\mathbb{E}[Y]-Y\right)^2\right] \\
&= \mathbb{E}\left[\left(\widehat{Y}-\mathbb{E}[Y]\right)^2\right] + \mathbb{E}\left[(\mathbb{E}[Y]-Y)^2\right] + 2\,\mathbb{E}\left[\left(\widehat{Y}-\mathbb{E}[Y]\right)(\mathbb{E}[Y]-Y)\right] \\
&= \mathbb{E}\left[\left(\mathbb{E}[Y]-\mathbb{E}\left[\widehat{Y}\right]+\mathbb{E}\left[\widehat{Y}\right]-\widehat{Y}\right)^2\right] + \mathrm{Var}(Y) \\
&= \left(\mathbb{E}[Y]-\mathbb{E}\left[\widehat{Y}\right]\right)^2 + \mathrm{Var}(\widehat{Y}) + \mathrm{Var}(Y).
\end{aligned}$$

This proves the proposition. □

Remarks 1.7 (generalization loss, part I)**.**

- The first term on the right-hand side of the statement of Proposition 1.6 denotes the squared *bias*, the second term the *estimation variance* and the last term the pure randomness (*process variance*) involved in the prediction problem. In general, we try to minimize simultaneously the bias and the estimation variance in order to get accurate predictions. Usually, these two terms compete in the sense that a decrease in one of them typically leads to an increase in the other one. This phenomenon is known as the bias-variance trade-off for which one needs to find a good balance (typically by controlling the complexity of the model). This is crucial for heterogeneous portfolios and it is going to be the main topic of these notes. We also refer to Section 7.3 in Hastie et al. [62] for the bias-variance trade-off.

- If we are in the atypical situation of having a homogeneous (in λ) Poisson portfolio and if we use the $\mathcal{D}$-based MLE $\widehat{\lambda}$, the situation becomes much more simple. In Section 1.2.2 we have seen that the MLE $\widehat{\lambda}$ is unbiased and we have determined its estimation variance. Henceforth, we receive in this special (simple) case for the MSEP generalization loss

$$\mathbb{E}\left[\left(Y-\widehat{Y}\right)^2\right] = \left(\mathbb{E}[Y]-\mathbb{E}\left[\widehat{\lambda}\right]\right)^2 + \mathrm{Var}(\widehat{\lambda}) + \mathrm{Var}(Y) = 0 + \frac{\lambda}{\sum_{i=1}^n v_i} + \frac{\lambda}{v}.$$

  We emphasize that this is an atypical situation because usually we do not assume to have a homogeneous portfolio and, in general, the MLE is not unbiased.

Proposition 1.6 considers the MSEP which implicitly implies that the weighted square loss function is the objective function of interest. However, in Section 1.2.2 we have been considering the Poisson deviance loss as objective function (to obtain the MLE) and, therefore, it is canonical to measure the generalization loss in terms of the *out-of-sample Poisson deviance loss*. Under the assumptions of Proposition 1.6 this means that we aim at studying

$$\mathbb{E}[D^*(N, \widehat{\lambda})] = 2\mathbb{E}\left[\widehat{\lambda}v - N - N\log(\widehat{\lambda}v/N)\right].$$

Proposition 1.8 (Poisson deviance generalization loss)**.** *Assume that all cases in $\mathcal{D}$ are independent and Poisson distributed having the same expected frequency $\lambda > 0$. Moreover, let case (N, v) be independent of $\mathcal{D}$ and Poisson distributed with the same expected frequency λ. We predict N by $\widehat{\lambda}v$, where the $\mathcal{D}$-based estimator $\widehat{\lambda}$ is assumed to be integrable and $\widehat{\lambda} \geq \epsilon$, $\mathbb{P}$-a.s., for some $\epsilon > 0$. This prediction has Poisson deviance generalization loss*

$$\mathbb{E}[D^*(N, \widehat{\lambda})] = \mathbb{E}[D^*(N, \lambda)] + \mathcal{E}(\widehat{\lambda}, \lambda),$$

with estimation loss defined by

$$\mathcal{E}(\widehat{\lambda}, \lambda) = 2v \left(\mathbb{E}\left[\widehat{\lambda}\right] - \lambda - \lambda \mathbb{E}\left[\log\left(\widehat{\lambda}/\lambda\right)\right]\right) \;\geq\; 0.$$

Proof of Proposition 1.8. The assumptions on $\widehat{\lambda}$ imply $\log \epsilon \leq \mathbb{E}[\log \widehat{\lambda}] \leq \log \mathbb{E}[\widehat{\lambda}] < \infty$. We have

$$\mathbb{E}[D^*(N, \widehat{\lambda})] = 2\mathbb{E}\left[\widehat{\lambda}v - \lambda v + \lambda v - N - N \log(\lambda v/N) - N \log(\widehat{\lambda}/\lambda)\right].$$

The first claims follows by the independence between N and $\widehat{\lambda}$. There remains the proof of the positivity of the estimation loss. Observe

$$\widehat{\lambda} - \lambda - \lambda \log\left(\widehat{\lambda}/\lambda\right) = \lambda\left(x - 1 - \log x\right) = \lambda g(x),$$

where we have defined $x = \widehat{\lambda}/\lambda > 0$, and the last identity defines the function $g(x) = x - 1 - \log x$. The claim now follows from $\log x \leq x - 1$ with that latter inequality being strict for $x \neq 1$. □

Remarks 1.9 (generalization loss, part II)**.**

- The estimation loss $\mathcal{E}(\widehat{\lambda}, \lambda)$ simultaneously quantifies the bias and the estimation volatility. Again, we try to make this term small by controlling the bias-variance trade-off.

- If we want to use the $\mathcal{D}$-based MLE $\widehat{\lambda}$ of Section 1.2.2 for estimating λ, we need to insure positivity, $\mathbb{P}$-a.s., i.e. we need to exclude degenerate models. We set $\widehat{\lambda}^\epsilon = \widehat{\lambda} + \epsilon$ for some $\epsilon > 0$. We receive Poisson deviance generalization loss for the MLE-based estimator $\widehat{\lambda}^\epsilon$

$$\mathbb{E}[D^*(N, \widehat{\lambda}^\epsilon)] = \mathbb{E}[D^*(N, \lambda)] + 2v\epsilon - 2v\lambda \mathbb{E}\left[\log\left(\widehat{\lambda}^\epsilon/\lambda\right)\right],$$

  with

$$\begin{aligned} \mathcal{E}(\widehat{\lambda}^\epsilon, \lambda) &= 2v\epsilon - 2\mathbb{E}[\lambda v \log(\widehat{\lambda}^\epsilon v/(\lambda v))] \;\geq\; 0, \\ \mathbb{E}[D^*(N, \lambda)] &= -2\mathbb{E}\left[N \log(\lambda v/N)\right] \;\geq\; 0. \end{aligned}$$

  Note that we may also use $\widehat{\lambda}^\epsilon = \max\{\widehat{\lambda}, \epsilon\}$ for guaranteeing positivity, $\mathbb{P}$-a.s., but in this case the bias is more difficult to determine.

- In view of the Poisson deviance generalization loss we can determine the optimal estimator for λ w.r.t. optimization criterion

$$\widehat{\lambda}^* = \arg\min_\mu \; \mathbb{E}[D^*(N, \mu)] = \arg\min_\mu \; 2\mathbb{E}[\mu v - N - N \log(\mu v/N)].$$

This minimizer is given by $\widehat{\lambda}^* = \mathbb{E}[N]/v$ which implies that the Poisson deviance loss is strictly consistent for the expected value according to Gneiting [55]; strict consistency is a property needed to perform back-testing, we refer to Section 4.1.3 in [141].

We close this section by comparing the square loss function, the Poisson deviance loss function and the absolute value loss function.

Example 1.10. In this example we illustrate the three different loss functions:

$$\begin{aligned} L(Y, \lambda, v=1) &= (Y-\lambda)^2, \\ L(Y, \lambda, v=1) &= 2\left[\lambda - Y - Y \log\left(\lambda/Y\right)\right], \\ L(Y, \lambda, v=1) &= |Y-\lambda|, \end{aligned}$$

the first one is the square loss function, the second one the Poisson deviance loss function and the third one the absolute value loss function.

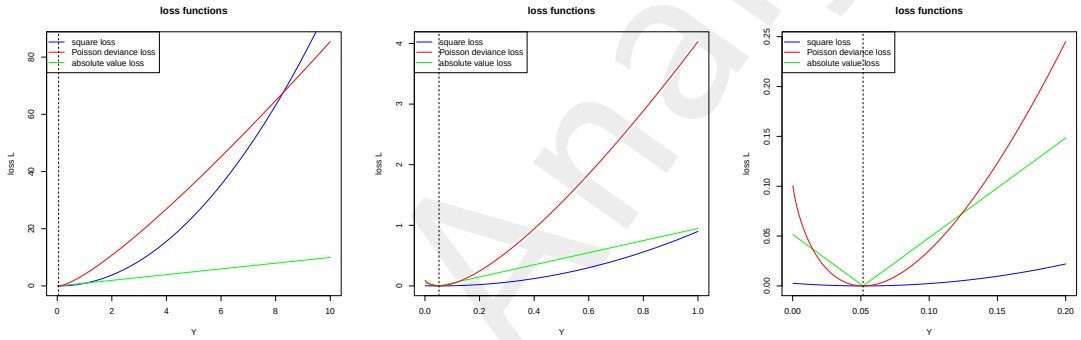


Figure 1.2: Loss functions $Y \mapsto L(Y, \lambda, v=1)$ with $\lambda = 5.16\%$ for different scales on the x-axis (and y-axis).

These three loss functions are plotted in Figure 1.2 for an expected frequency of $\lambda = 5.16\%$. From this plot we see that the pure randomness is (by far) the dominant term: the random variable $Y = N$ (for $v = 1$) lives on the integer values $\mathbb{N}_0$ and, thus, every positive outcome $Y \geq 1$ substantially contributes to the loss $L(\cdot)$, see Figure 1.2 (middle). A misspecification of λ only marginally contributes (for small λ); this is exactly the class imbalance problem that makes model calibration of low frequency examples so difficult. Moreover, we see that the Poisson deviance loss reacts more sensitively than the square loss function for claims $N \in \{1, \ldots, 8\}$ for our expected frequency choice of $\lambda = 5.16\%$. Formula (6.4) and Figure 6.2 in McCullagh–Nelder [93] propose the following approximation, we also refer to Figure 5.5 in [141],

$$L(Y, \lambda, v=1) \;\approx\; 9Y^{1/3}\left(Y^{1/3} - \lambda^{1/3}\right)^2.$$

Thus, Poisson deviance losses have a rather different behavior from square losses. ■

1.3.2 Cross-validation on test samples

Assessment of the quality of an estimated model is done by analyzing the generalization loss, see Propositions 1.6 and 1.8. That is, one back-tests the estimated model on cases that have not been seen during the estimation procedure. However, the true generalization loss cannot be determined because typically the true data generating mechanism is not known. Therefore, one tries to assess the generalization loss empirically.

Denote by $\widehat{\lambda} \mapsto L(Y, \widehat{\lambda}, v)$ a (non-negative) loss function, for examples see Example 1.10. The generalization loss for this loss function is defined by (subject to existence)

$$\mathcal{L}_L = \mathbb{E}\left[L(Y, \widehat{\lambda}, v)\right].$$

If the amount of data $\mathcal{D}$ in (1.6) is very large, we are tempted to partition the data $\mathcal{D}$ into a *learning sample* $\mathcal{D}^{\mathcal{B}}$ and a *test sample* $\mathcal{D}^{\mathcal{B}^c}$ with $\mathcal{B} \subset \{1, \ldots, n\}$ labeling the cases $\mathcal{D}^{\mathcal{B}} = \{(N_i, v_i);\ i \in \mathcal{B}\} \subset \mathcal{D}$ considered for learning the model, and with $\mathcal{B}^c = \{1, \ldots, n\} \setminus \mathcal{B}$ labeling the remaining test cases.

Based on the learning sample $\mathcal{D}^{\mathcal{B}}$ we construct the predictor $\widehat{\lambda}^{\mathcal{B}}$ for $Y = N/v$, see (1.7), and we use the test sample $\mathcal{D}^{\mathcal{B}^c}$ to estimate the generalization loss empirically by

$$\mathcal{L}_L^{\text{oos}} = \widehat{\mathbb{E}}\left[L(Y, \widehat{\lambda}, v)\right] = \frac{1}{|\mathcal{B}^c|} \sum_{i \in \mathcal{B}^c} L(Y_i, \widehat{\lambda}^{\mathcal{B}}, v_i). \tag{1.8}$$

The upper index in $\mathcal{L}_L^{\text{oos}}$ indicates that we do an *out-of-sample* analysis (estimation) because the data for estimation and back-testing has been partitioned into a learning sample and a disjoint back-testing sample.

The out-of-sample Poisson deviance loss is analogously estimated by, assume $\widehat{\lambda}^{\mathcal{B}} > 0$,

$$\mathcal{L}_D^{\text{oos}} = \widehat{\mathbb{E}}[D^*(N, \widehat{\lambda})] = \frac{1}{|\mathcal{B}^c|} \sum_{i \in \mathcal{B}^c} 2\, N_i \left[\frac{\widehat{\lambda}^{\mathcal{B}} v_i}{N_i} - 1 - \log\left(\frac{\widehat{\lambda}^{\mathcal{B}} v_i}{N_i}\right)\right] \ \geq\ 0. \tag{1.9}$$

In many situations it is too optimistic to assume that we can partition data $\mathcal{D}$ into a learning sample and a test sample because often the volume of the data is not sufficiently big. A naïve way to solve this problem is to use the whole data $\mathcal{D}$ for learning the model and then back-testing this model on the same data. The model estimated from the whole data $\mathcal{D}$ is denoted by $\widehat{\lambda}$ (assume $\widehat{\lambda} > 0$).

The in-sample Poisson deviance loss is defined by

$$\mathcal{L}_D^{\text{is}} \ =\ \frac{1}{n} D^*(\boldsymbol{N}, \widehat{\lambda}) \ =\ \frac{1}{n} \sum_{i=1}^{n} 2\, N_i \left[\frac{\widehat{\lambda} v_i}{N_i} - 1 - \log\left(\frac{\widehat{\lambda} v_i}{N_i}\right)\right], \tag{1.10}$$

i.e. this is exactly the empirical Poisson deviance loss of the estimated model.

This in-sample loss $\mathcal{L}_D^{\text{is}}$ is prone to over-fitting because it prefers more complex models that can follow observations more closely. However, this smaller in-sample loss does

not necessarily imply that a more detailed model generalizes better to unseen cases. Therefore, we need other evaluation methods. These are discussed next.

1.3.3 Leave-one-out cross-validation

Choose $i \in \{1, \ldots, n\}$ and consider partition $\mathcal{B}_i = \{1, \ldots, n\} \setminus \{i\}$ and $\mathcal{B}_i^c = \{i\}$. This provides on every $\mathcal{B}_i$ a predictor for $Y = N/v$ given by

$$\widehat{\lambda}^{(-i)} \stackrel{\text{def.}}{=} \widehat{\lambda}^{\mathcal{B}_i}.$$

The leave-one-out cross-validation loss for loss function $L(\cdot)$ is defined by

$$\mathcal{L}_L^{\text{loo}} = \frac{1}{n} \sum_{i=1}^{n} L\left(Y_i, \widehat{\lambda}^{(-i)}, v_i\right).$$

Leave-one-out cross-validation uses all data $\mathcal{D}$ for learning: the data $\mathcal{D}$ is split into a *training set* $\mathcal{D}^{\mathcal{B}_i}$ for (partial) learning and a *validation set* $\mathcal{D}^{\{i\}}$ for an out-of-sample validation iteratively for all $i \in \{1, \ldots, n\}$. Often the leave-one-out cross-validation is computationally too expensive as it requires fitting n times the model which is for large insurance portfolios too exhaustive.

1.3.4 K-fold cross-validation

For K-fold cross-validation we choose an integer $K \geq 2$ and partition $\{1, \ldots, n\}$ randomly into K disjoint subsets $\mathcal{B}_1, \ldots, \mathcal{B}_K$ of roughly the same size. This provides for every $k = 1, \ldots, K$ a training set

$$\mathcal{D}^{(-\mathcal{B}_k)} = \{(N_i, v_i);\ i \notin \mathcal{B}_k\} \subset \mathcal{D},$$

and the corresponding estimator for the expected frequency λ

$$\widehat{\lambda}^{(-\mathcal{B}_k)} \stackrel{\text{def.}}{=} \widehat{\lambda}^{\{1,\ldots,n\}\setminus\mathcal{B}_k}.$$

The K-fold cross-validation loss for loss function $L(\cdot)$ is defined by

$$\mathcal{L}_L^{\text{CV}} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in \mathcal{B}_k} L\left(Y_i, \widehat{\lambda}^{(-\mathcal{B}_k)}, v_i\right) \approx \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} L\left(Y_i, \widehat{\lambda}^{(-\mathcal{B}_k)}, v_i\right).$$

Note that we use the whole data $\mathcal{D}$ for learning. As for leave-one-out cross-validation we split this learning data into a training set $\mathcal{D}^{(-\mathcal{B}_k)}$ for (partial) learning and a validation set $\mathcal{D}^{\mathcal{B}_k}$ for out-of-sample validation. This is done for all $k = 1, \ldots, K$, and the out-of-sample (generalization) loss is then estimated by the resulting average cross-validation loss. K-fold cross-validation is the method typically used, and in many applications one chooses $K = 10$. We do not further elaborate on this choice here, but we refer to the related literature.

1.3.5 Stratified K-fold cross-validation

K-fold cross-validation partitions $\{1,\ldots,n\}$ into K disjoint random subsets $\mathcal{B}_1,\ldots,\mathcal{B}_K$ of approximately the same size. If there are outliers then these outliers may fall into the same subset $\mathcal{B}_k$, and this may disturb K-fold cross-validation results
Stratified K-fold cross-validation distributes outliers more equally across the partition. This is achieved by ordering the observations $Y_i = N_i/v_i$, $i = 1,\ldots,n$, that is, $Y_{(1)} \geq Y_{(2)} \geq \ldots \geq Y_{(n)}$, with a deterministic rule if there is more than one observation of the same size. Then we build urns $\mathcal{U}_j$ of size K for $j = 1,\ldots,\lceil n/K \rceil$ (the last urn $\mathcal{U}_{\lceil n/K \rceil}$ may be smaller depending on the cardinality of n and K)

$$\mathcal{U}_j = \left\{ Y_{(i)};\ (j-1)K + 1 \leq i \leq jK \right\}.$$

Urn $\mathcal{U}_1$ receives the K largest observations, urn $\mathcal{U}_2$ contains the next K largest observations, etc. Then we define the partition $(\mathcal{D}_k)_{k=1,\ldots,K}$ of $\mathcal{D}$ for $k = 1,\ldots,K$ by

$$\mathcal{D}_k = \left\{ \text{choose randomly from each urn } \mathcal{U}_1,\ldots,\mathcal{U}_{\lceil n/K \rceil} \text{ one case (without replacement)} \right\},$$

where "choose randomly (without replacement)" is meant in the sense that all urns are randomly distributed resulting in the partitioned data $\mathcal{D}_k$, $k = 1,\ldots,K$.
K-fold cross-validation is now applied to the partition $\mathcal{D}_1,\ldots,\mathcal{D}_K$. Note that this does not necessarily provide the same result as the original K-fold cross-validation because in the stratified version it is impossible that the two largest outliers fall into the same set of the partition (supposed that they are bigger than the remaining observations).

Summary. To estimate the generalization loss we typically choose the (stratified) K-fold Poisson deviance cross-validation loss given by

$$\mathcal{L}_D^{\mathrm{CV}} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in \mathcal{B}_k} 2 \left[\widehat{\lambda}^{(-\mathcal{B}_k)} v_i - N_i - N_i \log \left(\frac{\widehat{\lambda}^{(-\mathcal{B}_k)} v_i}{N_i} \right) \right] \ \geq \ 0. \tag{1.11}$$

Remark. Evaluation of the K-fold Poisson deviance cross-validation loss (1.11) requires that the (random) partition $\mathcal{B}_1,\ldots,\mathcal{B}_K$ of $\mathcal{D}$ is chosen such that $\widehat{\lambda}^{(-\mathcal{B}_k)} > 0$ for all $k = 1,\ldots,K$. This is guaranteed in stratified K-fold cross-validation as soon as we have K observations with $N_i > 0$. For non-stratified K-fold cross-validation the situation is more complicated because the positivity constraint may fail with positive probability (if we have too many claims with $N_i = 0$).

1.4 Example: homogeneous Poisson model

Throughout these notes we consider a synthetic motor-third party liability (MTPL) car insurance portfolio. This portfolio consists of $n = 500'000$ car insurance policies having claims N_i information and years at risk information $v_i \in (0,1]$, for $i = 1,\ldots,n$. The simulation of the data is described in Appendix A, in particular, we refer to Listing A.2 which sketches this data $\mathcal{D}$.[4] In a first (homogeneous) statistical model we assume

$$N_i \overset{\text{ind.}}{\sim} \mathrm{Poi}(\lambda v_i) \qquad \text{for } i = 1,\ldots,n,$$

[4] The data is available from `https://people.math.ethz.ch/~wmario/Lecture/MTPL_data.csv`

and with given expected frequency $\lambda > 0$. The MLE for λ of this data $\mathcal{D}$ is given by, see (1.3),

$$\widehat{\lambda} = \frac{\sum_{i=1}^n N_i}{\sum_{i=1}^n v_i} = 10.2691\%.$$

This should be compared to the true average frequency $\bar{\lambda}^\star = 10.1991\%$ given in (A.4). Thus, we have a small positive bias in our estimate. The in-sample Poisson deviance loss is given by $\mathcal{L}_D^{\text{is}} = 29.1065 \cdot 10^{-2}$.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%

Table 1.1: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters.

Next, we calculate the 10-fold cross-validation losses (1.11). The results are presented in columns $\mathcal{L}_D^{\text{CV}}$ (non-stratified and stratified versions) of Table 1.1. We observe that this 10-fold cross-validation losses of $29.1066 \cdot 10^{-2}$ and $29.1065 \cdot 10^{-2}$, respectively, match the in-sample loss $\mathcal{L}_D^{\text{is}} = 29.1065 \cdot 10^{-2}$, i.e. we do not have any sign of over-fitting, here. The column 'run time' shows the total run time needed,[5] and '# param.' gives the number of estimated model parameters.

Finally, we determine the estimation loss $\mathcal{E}(\widehat{\lambda}, \lambda^\star)$ w.r.t. the true model $\lambda^\star$, see Appendix A and Proposition 1.8. We emphasize that we are in the special situation here of knowing the true model $\lambda^\star$ because we work with synthetic data. This is an atypical situation in practice and therefore we highlight all values in green color which can only be calculated because we know the true model. Using in-sample loss (1.10) we derive

$$\begin{aligned}\mathcal{L}_D^{\text{is}} &= \frac{1}{n}\sum_{i=1}^n 2\, N_i \left[\frac{\lambda^\star(\boldsymbol{x}_i)v_i}{N_i} - \frac{\lambda^\star(\boldsymbol{x}_i)v_i}{N_i} + \frac{\widehat{\lambda}v_i}{N_i} - 1 - \log\left(\frac{\lambda^\star(\boldsymbol{x}_i)v_i}{N_i}\right) - \log\left(\frac{\widehat{\lambda}}{\lambda^\star(\boldsymbol{x}_i)}\right)\right] \\ &= \frac{1}{n} D^*(\boldsymbol{N}, \lambda^\star) + \widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star) + \frac{1}{n}\sum_{i=1}^n 2\left(\lambda^\star(\boldsymbol{x}_i)v_i - N_i\right)\log\left(\frac{\widehat{\lambda}}{\lambda^\star(\boldsymbol{x}_i)}\right), \qquad (1.12)\end{aligned}$$

where the first term is the Poisson deviance loss w.r.t. the true model $\lambda^\star$, see (A.5),

$$\frac{1}{n} D^*(\boldsymbol{N}, \lambda^\star) = \frac{1}{n}\sum_{i=1}^n 2\left[\lambda^\star(\boldsymbol{x}_i)v_i - N_i - N_i \log\left(\frac{\lambda^\star(\boldsymbol{x}_i)v_i}{N_i}\right)\right] = 27.7278 \cdot 10^{-2}.$$

The second term in (1.12) is defined by

$$\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star) = \frac{1}{n}\sum_{i=1}^n 2v_i \left[\widehat{\lambda} - \lambda^\star(\boldsymbol{x}_i) - \lambda^\star(\boldsymbol{x}_i)\log\left(\frac{\widehat{\lambda}}{\lambda^\star(\boldsymbol{x}_i)}\right)\right] = 1.3439 \cdot 10^{-2} \;\geq\; 0. \qquad (1.13)$$

[5] All run times are measured on a personal laptop Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz with 16GB RAM.

This is an estimate for the estimation loss $\mathcal{E}(\widehat{\lambda}, \lambda^\star)$ w.r.t. the true model $\lambda^\star$.[6] The last term in (1.12) refers to over-fitting, in average it disappear if the estimation $\widehat{\lambda}$ is independent from $\boldsymbol{N}$. In our case it takes value

$$\frac{1}{n} \sum_{i=1}^{n} 2 \left(\lambda^\star(\boldsymbol{x}_i) v_i - N_i\right) \log\left(\frac{\widehat{\lambda}}{\lambda^\star(\boldsymbol{x}_i)}\right) = 0.0348 \cdot 10^{-2},$$

i.e. it is negligible w.r.t. the other terms.
We will use the estimated estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ as a measure of accuracy in all models considered, below. Note that we have $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star) = 0$ if and only if $\widehat{\lambda} = \lambda^\star(\boldsymbol{x}_i)$ for all $i = 1, \ldots, n$. Thus, obviously, if we have heterogeneity between the expected frequencies of the insurance policies, the estimation loss cannot be zero for a homogeneous model. ■

Summary. In practice, model assessment is done w.r.t. cross-validation losses, see Table 1.1. In our special situation of knowing the true model and the true expected frequency function $\lambda^\star(\cdot)$, we will use the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ for model assessment. This also allows us to check whether we draw the right conclusions based on the cross-validation analysis.
In mathematical statistics, the estimation loss (1.13) is related to the *risk* stemming from a *decision rule*. In our situation the decision rule is the MLE $\boldsymbol{N} \mapsto \widehat{\lambda} = \widehat{\lambda}(\boldsymbol{N})$ which is compared to the true parameters $(\lambda^\star(\boldsymbol{x}_i))_{i=1,\ldots,n}$ in terms of the loss function under the summation in (1.13).

[6] Strictly speaking we should consider the estimation loss $\mathcal{E}(\widehat{\lambda}^\epsilon, \lambda^\star)$ of $\widehat{\lambda}^\epsilon = \max\{\widehat{\lambda}, \epsilon\}$ for some $\epsilon > 0$, because we need a strictly positive estimator, $\mathbb{P}$-a.s., see Proposition 1.8.

Chapter 2

Generalized Linear Models

Generalized linear models (GLMs) are popular statistical models that are used in the framework of the exponential dispersion family (EDF). For standard references on GLMs and the EDF we refer to Nelder–Wedderburn [98] and McCullagh–Nelder [93]. We present the Poisson claims count model within a GLM framework in this chapter. This extends the homogeneous Poisson portfolio consideration of Section 1.2.2 to the heterogeneous case. Similar derivations can be done for individual claim sizes using, for instance, a gamma or a log-normal distribution for individual claim sizes, for details we refer to Ohlsson–Johansson [102] and Wüthrich–Merz [141]. Additionally, we introduce a classification problem in this chapter which is based on the Bernoulli model and which describes a logistic regression problem, see Section 2.5 below.

2.1 Heterogeneous Poisson claims frequency model

Assume that the claims count random variable N has a Poisson distribution with given years at risk $v > 0$ and expected frequency $\lambda > 0$. We aim at modeling the expected frequency $\lambda > 0$ such that it allows us to incorporate structural differences (heterogeneity) between different insurance policies and risks; such structural differences are called systematic effects in the statistical literature. Assume we have q-dimensional features $\boldsymbol{x} = (x_1, \ldots, x_q)' \in \mathcal{X}$ belonging to the set of all possible features (called feature space $\mathcal{X}$). A regression function $\lambda(\cdot)$ maps this feature space $\mathcal{X}$ to expected frequencies

$$\lambda : \mathcal{X} \to \mathbb{R}_+, \qquad \boldsymbol{x} \mapsto \lambda = \lambda(\boldsymbol{x}). \tag{2.1}$$

The feature $\boldsymbol{x}$ describes the risk characteristics of a certain insurance policy, see Example 2.1 below, and $\lambda(\cdot)$ describes the resulting structural differences (systematic effects) in the expected frequency described by these features.

Terminology. There is different terminology and we use the ones in *italic letters*.

- $\boldsymbol{x}$ is called *feature*, covariate, explanatory variable, independent variable, predictor variable, measurement vector.
- λ is called (expected) *response*, dependent variable, regressand, *regression function*, *classifier* (the latter three terminologies also depend on the particular type of

regression problem considered, this is further highlighted below).

- The components of $\boldsymbol{x}$ can be continuous, discrete or even finite. In the finite and discrete cases we distinguish between *ordered* (*ordinal*) and *unordered* (*nominal*) components (called *categorical* components). For instance, $x_l \in \{\text{female}, \text{male}\}$ is a nominal categorical feature component. The case of two categories is called binary.

Our main goal is to find the regression function $\lambda(\cdot)$ as a function of the features $\boldsymbol{x} \in \mathcal{X}$ and to understand the systematic effects and their sensitivities in each feature component x_l of $\boldsymbol{x}$. In general, available observations are noisy, that is, we cannot directly observe $\lambda(\boldsymbol{x})$, but only the frequency $Y = N/v$, which is generated by

$$Y = \lambda(\boldsymbol{x}) + \varepsilon,$$

with residual (noise) term ε satisfying in the Poisson case

$$\mathbb{E}[\varepsilon] = 0 \qquad \text{and} \qquad \mathrm{Var}(\varepsilon) = \lambda(\boldsymbol{x})/v.$$

Example 2.1 (car insurance)**.** We model the claims frequency of $Y = N/v$ with feature $\boldsymbol{x}$ by

$$\mathbb{P}\left[Y = k/v\right] \;=\; \mathbb{P}\left[N = k\right] \;=\; \exp\left\{-\lambda(\boldsymbol{x})v\right\} \, \frac{(\lambda(\boldsymbol{x})v)^k}{k!} \qquad \text{for } k \in \mathbb{N}_0,$$

with given years at risk $v > 0$ and regression function $\lambda(\cdot)$ given by (2.1). We may now think of features $\boldsymbol{x} = (x_1, \ldots, x_q)$ characterizing different car drivers with, for instance, x_1 describing the age of the driver (continuous component), x_2 describing the price of the car (continuous component or discrete ordinal component if of type "cheap", "average", "expensive"), x_3 describing the gender of the driver (binary component), etc. The goal is to find (infer) the regression function $\lambda(\cdot)$ so that it optimally characterizes the underlying risks in terms of the chosen features $\boldsymbol{x} \in \mathcal{X}$. This inference needs to be done from noisy claims count observations N and the chosen features may serve as *proxies* for other (unobservable) risk drivers such as driving experience, driving skills, etc. ■

In view of the previous example it seems advantageous to include as many feature components as possible in the model. However, if the feature space is too complex and too large this will lead to a poor model with a poor predictive performance (big generalization loss, see Section 1.3). The reason for this being that we have to infer the regression function from a *finite* number of observations, and the bigger the feature space the more likely that irrelevant feature information may play a special role in a particular (finite) observation sample. For this reason it is important to carefully choose relevant feature information. In fact, feature extraction is one of the best studied and understood fields in actuarial practice with a long tradition. The Swiss car insurance market has been deregulated in 1996 and since then sophisticated pricing models have been developed that may depend on more than 30 feature components. One may ask, for instance, questions like "What is a sports car?", see Ingenbleek–Lemaire [72].

2.2 Multiplicative regression model

A commonly used technique to infer a regression function $\lambda(\cdot)$ is the MLE method within the family of GLMs. We consider the special case of the Poisson model with a *multiplicative* regression structure in this chapter. This multiplicative regression structure leads to a *log-linear* functional form (or the log-link, respectively).
Assume $\mathcal{X} \subset \mathbb{R}^q$ and that the regression function $\lambda(\cdot)$ is characterized by a parameter vector $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_q)' \in \mathbb{R}^{q+1}$ such that for $\boldsymbol{x} \in \mathcal{X}$ we have representation

$$\boldsymbol{x} \;\mapsto\; \log \lambda(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_q x_q \;\stackrel{\text{def.}}{=}\; \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle. \tag{2.2}$$

The last definition uses a slight abuse of notation because the features $\boldsymbol{x} \in \mathcal{X}$ are extended by a zero component $x_0 \equiv 1$ for the intercept component β_0. Formula (2.2) assumes that all feature components are real-valued. This requires that categorical feature components are transformed (pre-processed); a detailed treatment and description of categorical feature components' pre-processing is provided in Section 2.4.1, below.
The task is to infer $\boldsymbol{\beta}$ from cases $(N_i, \boldsymbol{x}_i, v_i) \in \mathcal{D}$, where we extend definition (1.6) of the data to

$$\mathcal{D} = \{(N_1, \boldsymbol{x}_1, v_1), \ldots, (N_n, \boldsymbol{x}_n, v_n)\}, \tag{2.3}$$

with $\boldsymbol{x}_i \in \mathcal{X}$ being the feature information of policy $i = 1, \ldots, n$. Assume that all cases are *independent* with N_i being Poisson distributed with expected frequency $\lambda(\boldsymbol{x}_i)$ given by (2.2). The joint log-likelihood function of the data $\mathcal{D}$ under these assumptions is given by

$$\boldsymbol{\beta} \;\mapsto\; \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} -\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle v_i + N_i \left(\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle + \log v_i \right) - \log(N_i!). \tag{2.4}$$

The MLE may be found by the solution of[1]

$$\frac{\partial}{\partial \boldsymbol{\beta}}\, \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = 0. \tag{2.5}$$

We calculate the partial derivatives of the log-likelihood function for $0 \le l \le q$

$$\frac{\partial}{\partial \beta_l} \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} -\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle v_i x_{i,l} + N_i x_{i,l} \;=\; \sum_{i=1}^{n} \left(-\lambda(\boldsymbol{x}_i) v_i + N_i \right) x_{i,l} \;=\; 0, \tag{2.6}$$

where $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,q})' \in \mathcal{X}$ describes the feature of the i-th case in $\mathcal{D}$, and for the intercept β_0 we add components $x_{i,0} = 1$. We define the design matrix $\mathfrak{X} \in \mathbb{R}^{n \times (q+1)}$ by

$$\mathfrak{X} = (x_{i,l})_{1 \le i \le n, 0 \le l \le q}, \tag{2.7}$$

thus, each row $1 \le i \le n$ describes the feature $\boldsymbol{x}_i'$ of case i. Observe that $(\mathfrak{X}\boldsymbol{\beta})_i = \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = \log \lambda(\boldsymbol{x}_i)$ and $(\mathfrak{X}'\boldsymbol{N})_l = \sum_{i=1}^{n} N_i x_{i,l}$. This allows us to rewrite (2.6) as follows

$$(\mathfrak{X}'\boldsymbol{N})_l = \sum_{i=1}^{n} \exp\left\{ (\mathfrak{X}\boldsymbol{\beta})_i \right\} v_i x_{i,l} = \left(\mathfrak{X}' V \exp\left\{ \mathfrak{X}\boldsymbol{\beta} \right\} \right)_l,$$

[1] Formula (2.5) is a bit a sloppy notation of saying that the gradient $\nabla_{\boldsymbol{\beta}} \ell_{\boldsymbol{N}}(\boldsymbol{\beta})$ is equal to the zero vector in $\mathbb{R}^{q+1}$. Solutions to (2.5) give critical points, maximas have negative definite Hessians, and in case of a concave log-likelihood function we have a unique maximum.

where we define the weight matrix $V = \text{diag}(v_1, \ldots, v_n)$ and where the exponential function $\exp\{\mathfrak{X}\boldsymbol{\beta}\}$ is understood element-wise. We have just derived the following statement:

Proposition 2.2. *The solution $\widehat{\boldsymbol{\beta}}$ to the MLE problem* (2.5) *in the Poisson case* (2.4) *may be found by the solution of*

$$\mathfrak{X}' V \exp\{\mathfrak{X}\boldsymbol{\beta}\} = \mathfrak{X}' \boldsymbol{N}.$$

This optimization problem in Proposition 2.2 is solved numerically using Fisher's scoring method or the iteratively re-weighted least squares (IRLS) method, see Nelder–Wedderburn [98]. These algorithms are versions of the Newton–Raphson algorithm to find roots of sufficiently smooth functions.

This MLE $\widehat{\boldsymbol{\beta}}$ is then used to estimate the regression function $\lambda(\cdot)$ and we set for $\boldsymbol{x} \in \mathcal{X}$

$$\boldsymbol{x} \;\mapsto\; \widehat{\lambda}(\boldsymbol{x}) = \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle, \tag{2.8}$$

which provides the predictor for case $(Y, \boldsymbol{x}, v)$ in the Poisson GLM situation

$$\widehat{Y} = \widehat{\mathbb{E}}[Y] = \widehat{\lambda}(\boldsymbol{x}) = \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle.$$

Remarks 2.3.

- The Hessian of $\ell_{\boldsymbol{N}}(\boldsymbol{\beta})$ is given by

$$\mathbf{H}_{\boldsymbol{\beta}} \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = -\mathfrak{X}' \,\text{diag}\left(\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_1 \rangle v_1, \ldots, \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_n \rangle v_n\right) \mathfrak{X} \;\in\; \mathbb{R}^{(q+1)\times(q+1)}. \tag{2.9}$$

  Below we make assumptions on the rank of the design matrix $\mathfrak{X}$, which is necessary to have uniqueness of the MLE in Proposition 2.2. In fact, if $\mathfrak{X}$ has full rank $q+1 \leq n$ then we have a negative definite Hessian $\mathbf{H}_{\boldsymbol{\beta}} \ell_{\boldsymbol{N}}(\boldsymbol{\beta})$ which provides a unique MLE. This follows from the property that (2.4) provides a concave optimization problem in a minimal representation if $\mathfrak{X}$ has full rank.

- In the derivation of the MLE (2.5) we assume that we know the true functional form for $\lambda(\cdot)$ of the data generating mechanism and only its parameter $\boldsymbol{\beta}$ needs to be inferred, see (2.2). Typically, the true data generating mechanism is not known and (2.8) is used as an approximation to the true (but unknown) mechanism. In the next chapters we present methods that assume less model structure in $\lambda(\cdot)$.

- The log-linear structure (2.2) for the expected frequency implies a multiplicative tariff structure (systematic effects)

$$\widehat{Y} = \widehat{\mathbb{E}}[Y] = \widehat{\lambda}(\boldsymbol{x}) = \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle = \exp\{\widehat{\beta}_0\} \prod_{l=1}^{q} \exp\{\widehat{\beta}_l x_l\}. \tag{2.10}$$

  The term $\exp\{\widehat{\beta}_0\}$ from the intercept $\widehat{\beta}_0$ describes the base premium and the factors $\exp\{\widehat{\beta}_l x_l\}$ describe the adjustments according to the feature components x_l. These factors $\exp\{\widehat{\beta}_l x_l\}$ are typically around 1 (if $\widehat{\beta}_0$ is appropriately normalized). Moreover, they can easily be interpreted. For instance, increasing feature component

x_l from value 1 to value 2 adds an additional relative loading of size $\exp\{\widehat{\beta}_l\}$ to the price; and this relative loading is independent of any other feature component values.

- Formula (2.10) shows that the feature components interact in a multiplicative way in our Poisson GLM. One of the main tasks below is to analyze whether this multiplicative interaction is appropriate.

- The so-called *score* is obtained by the gradient of the log-likelihood function

$$s(\boldsymbol{\beta}, \boldsymbol{N}) = \frac{\partial}{\partial \boldsymbol{\beta}} \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = \mathfrak{X}' \boldsymbol{N} - \mathfrak{X}' V \exp\{\mathfrak{X}\boldsymbol{\beta}\}.$$

Under the assumption that the data $\mathcal{D}$ has been generated by this Poisson GLM we have $\mathbb{E}[s(\boldsymbol{\beta}, \boldsymbol{N})] = 0$ and we obtain *Fisher's information* $\mathcal{I}(\boldsymbol{\beta}) \in \mathbb{R}^{(q+1)\times(q+1)}$, see also Section 2.6 in the appendix,

$$\mathcal{I}(\boldsymbol{\beta}) = \mathbb{E}\left[s(\boldsymbol{\beta}, \boldsymbol{N}) s(\boldsymbol{\beta}, \boldsymbol{N})'\right] = -\mathbb{E}\left[\frac{\partial}{\partial \boldsymbol{\beta}} s(\boldsymbol{\beta}, \boldsymbol{N})\right] = -\mathbf{H}_{\boldsymbol{\beta}} \ell_{\boldsymbol{N}}(\boldsymbol{\beta}).$$

From Proposition 2.2 we obtain unbiasedness of the volume-adjusted MLE (2.8) for the expected number of claims, see also Proposition 2.4, below. Moreover, the Cramér–Rao bound is attained, which means that we have a uniformly minimum variance unbiased (UMVU) estimator, see Section 2.7 in Lehmann [87].

Fisher's information matrix $\mathcal{I}(\boldsymbol{\beta})$ plays a crucial role in uncertainty quantification of MLEs within GLMs. In fact, one can prove asymptotic normality of the MLE $\widehat{\boldsymbol{\beta}}$ if the volumes go to infinity and the asymptotic covariance matrix is a scaled version of the inverse of Fisher's information matrix, we refer to Chapter 6 in Lehmann [87], Fahrmeir–Tutz [41] and Chapter 5 in Wüthrich–Merz [141].

Finally, we consider the so-called *balance property*, see Theorem 4.5 in Bühlmann-Gisler [18]. This is an important property in insurance to receive the right price calibration on the portfolio level.

Proposition 2.4 (balance property)**.** *Under the assumptions of Proposition 2.2 we have for the MLE* $\widehat{\boldsymbol{\beta}}$

$$\sum_{i=1}^n v_i \widehat{\lambda}(\boldsymbol{x}_i) = \sum_{i=1}^n v_i \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle = \sum_{i=1}^n N_i.$$

Proof. The proof is a direct consequence of Proposition 2.2. Note that the first column in the design matrix $\mathfrak{X}$ is identically equal to 1 (modeling the intercept). This implies

$$\sum_{i=1}^n v_i \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle = (1, \ldots, 1) V \exp\{\mathfrak{X}\boldsymbol{\beta}\} = (1, \ldots, 1)\boldsymbol{N} = \sum_{i=1}^n N_i.$$

This proves the claim. □

Remark 2.5. The balance property holds true in general for GLMs within the EDF as long as we work with the canonical link. The canonical link of the Poisson model is the log-link, which exactly tells us that for regression function (2.2) the balance property has to hold. For more background on canonical links we refer to McCullagh–Nelder [93] and Section 5.1.5 in [141].

2.3 Deviance residuals and parameter reduction

There are several ways to assess goodness of fit and to evaluate the generalization loss introduced in Section 1.3. For analyzing the generalization loss we consider the (stratified) K-fold Poisson deviance cross-validation loss described in (1.11), modified to the heterogeneous case. The (heterogeneous) Poisson deviance loss for regression function (2.2) is given by, see also (1.5),

$$D^*(\boldsymbol{N}, \lambda) = \sum_{i=1}^{n} 2\ N_i \left[\frac{\lambda(\boldsymbol{x}_i)v_i}{N_i} - 1 - \log\left(\frac{\lambda(\boldsymbol{x}_i)v_i}{N_i} \right) \right] \ \geq\ 0. \tag{2.11}$$

Note, again, that minimizing this Poisson deviance loss provides the MLE $\widehat{\boldsymbol{\beta}}$ under assumption (2.2).

For the goodness of fit we may consider the (in-sample) Pearson's residuals. We set $Y_i = N_i/v_i$ for $1 \leq i \leq n$

$$\widehat{\delta}_i^P = \frac{N_i - \widehat{\lambda}(\boldsymbol{x}_i)v_i}{\sqrt{\widehat{\lambda}(\boldsymbol{x}_i)v_i}} = \sqrt{v_i}\ \frac{Y_i - \widehat{\lambda}(\boldsymbol{x}_i)}{\sqrt{\widehat{\lambda}(\boldsymbol{x}_i)}}. \tag{2.12}$$

These Pearson's residuals should roughly be centered with unit variance (and close to independence) under our model assumptions. Therefore, we can consider scatter plots of these residuals (in relation to their features) and we should not detect any structure in these scatter plots.

Pearson's residuals $\widehat{\delta}_i^P$, $1 \leq i \leq n$, are distribution-free, i.e. they do not (directly) account for a particular distributional form. They are most appropriate in a (symmetric) Gaussian case. For other distributional models one often prefers deviance residuals. The reason for this preference is that deviance residuals are more robust (under the right distributional choice): note that the expected frequency parameter estimate $\widehat{\lambda}$ appears in the denominator of Pearson's residuals in (2.12). This may essentially distort Pearson's residuals. Therefore, we may not want to rely on weighted residuals. Moreover, Pearson's residuals do not account for the distributional properties of the underlying model. Therefore, Pearson's residuals can be heavily distorted by skweness and extreme observations (which look very non-Gaussian).

The Poisson deviance residuals are defined by

$$\widehat{\delta}_i^D = \operatorname{sgn}\left(N_i - \widehat{\lambda}(\boldsymbol{x}_i)v_i\right) \sqrt{2\ N_i \left[\frac{\widehat{\lambda}(\boldsymbol{x}_i)v_i}{N_i} - 1 - \log\left(\frac{\widehat{\lambda}(\boldsymbol{x}_i)v_i}{N_i} \right) \right]}. \tag{2.13}$$

Remarks.

- If we allow for an individual expected frequency parameter λ_i for each observation N_i, then the MLE optimal model is exactly the saturated model with parameter estimates $\widehat{\lambda}_i = N_i/v_i$, see also (1.4). Therefore, each term in the summation on the right-hand side of the above deviance loss (2.11) is non-negative, i.e.

$$2\ N_i \left[\frac{\widehat{\lambda}(\boldsymbol{x}_i)v_i}{N_i} - 1 - \log\left(\frac{\widehat{\lambda}(\boldsymbol{x}_i)v_i}{N_i} \right) \right] \geq 0,$$

for all $1 \leq i \leq n$. This implies that the deviance loss is bounded from below by zero, and a sequence of parameters $(\boldsymbol{\beta}_t)_{t\geq 1}$ that is decreasing in the sequence of deviance losses $(D^*(\boldsymbol{N}, \lambda = \lambda_{\boldsymbol{\beta}_t}))_{t\geq 1}$ may provide convergence to a (local) minimum w.r.t. that loss function. This is important in any numerical optimization such as the gradient descent algorithm that we will meet below.

- Observe that maximizing the log-likelihood $\ell_{\boldsymbol{N}}(\boldsymbol{\beta})$ for parameter $\boldsymbol{\beta}$ is equivalent to minimizing the deviance loss $D^*(\boldsymbol{N}, \lambda = \lambda_{\boldsymbol{\beta}})$ for $\boldsymbol{\beta}$. In this spirit, the deviance loss plays the role of the canonical *objective function* that should be minimized.

- $D^*(\boldsymbol{N}, \widehat{\lambda})$ is the *scaled* Poisson deviance loss. Within the EDF there is a second deviance statistics that accounts for potential over- or under-dispersion $\phi \neq 1$. In the Poisson model this does not apply because by definition $\phi = 1$. Nevertheless, we can determine this dispersion parameter empirically on our data. There are two different estimators. Pearson's (distribution-free) dispersion estimate is given by

$$\widehat{\phi}_P = \frac{1}{n-(q+1)} \sum_{i=1}^{n} \left(\widehat{\delta}_i^P\right)^2,$$

and the deviance dispersion estimate is given by

$$\widehat{\phi}_D = \frac{1}{n-(q+1)} \sum_{i=1}^{n} \left(\widehat{\delta}_i^D\right)^2 = \frac{D^*(\boldsymbol{N}, \widehat{\lambda})}{n-(q+1)}. \tag{2.14}$$

Pearson's dispersion estimate should be roughly equal to 1 to support our model choice. The size of the deviance dispersion estimate depends on the size of the expected number of claims λv, see Figure 1.1. For our true expected frequency $\lambda^\star$ we receive $\widehat{\phi}_D = D^*(\boldsymbol{N}, \lambda^\star)/n = 27.7228 \cdot 10^{-2}$, see (A.5).

Finally, we would like to test whether we need all components in $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$ or whether a lower dimensional *nested model* can equally well explain the observations $\boldsymbol{N}$. We assume that the components in $\boldsymbol{\beta}$ are ordered in the appropriate way, otherwise we permute them (and accordingly the columns of the design matrix $\mathfrak{X}$).

Null hypothesis H_0: $\beta_1 = \ldots = \beta_p = 0$ for given $1 \leq p \leq q$.

1. Calculate the deviance loss $D^*(\boldsymbol{N}, \widehat{\lambda}_{\text{full}})$ in the full model with MLE $\widehat{\boldsymbol{\beta}} \in \mathbb{R}^{q+1}$.
2. Calculate the deviance loss $D^*(\boldsymbol{N}, \widehat{\lambda}_{H_0})$ under the null hypothesis H_0 with MLE $\widehat{\boldsymbol{\beta}} \in \mathbb{R}^{q+1-p}$.

Define the likelihood ratio test statistics, see Lemma 3.1 in Ohlsson–Johansson [102],

$$\chi^2_{\mathcal{D}} = D^*(\boldsymbol{N}, \widehat{\lambda}_{H_0}) - D^*(\boldsymbol{N}, \widehat{\lambda}_{\text{full}}) \geq 0. \tag{2.15}$$

Under H_0, the likelihood ratio test statistics $\chi^2_{\mathcal{D}}$ is approximately χ^2-distributed with p degrees of freedom. Alternatively, one could use a Wald statistics instead of $\chi^2_{\mathcal{D}}$. A Wald statistics is a second order approximation to the log-likelihood function which is motivated by asymptotic normality of the MLE $\widehat{\boldsymbol{\beta}}$. The z-test in the R output in Listing 2.2 refers to a rooted Wald statistics; for more details we refer to Section 5.3.2 in [141].

Remarks 2.6.

- Analogously, the likelihood ratio test and the Wald test can be applied recursively to a sequence of nested models. This leads to a step-wise reduction of model complexity, this is similar in spirit to the analysis of variance (ANOVA) in Listing 2.7, and it is often referred to as backward model selection, see `drop1` below.
- The tests presented apply to nested models. If we want to selected between non-nested models we can use cross-validation.

2.4 Example in car insurance pricing

We consider a car insurance example with synthetic data $\mathcal{D}$ generated as described in Appendix A. The portfolio consists of $n = 500'000$ car insurance policies for which we have feature information $\boldsymbol{x}_i \in \mathcal{X}^\star$ and years at risk information $v_i \in (0, 1]$, for $i = 1, \ldots, n$. Moreover, for each policy i we have an observation N_i. These (noisy) observations were generated from independent Poisson distributions based on the underlying expected frequency function $\lambda^\star(\cdot)$ as described in Appendix A.[2] Here, we assume that neither the relevant feature components, the functional form nor the involved parameters of this expected frequency function $\lambda^\star(\cdot)$ are known (this is the typical situation in practice), and we aim at inferring an expected frequency function estimate $\widehat{\lambda}(\cdot)$ from the data $\mathcal{D}$.[3]

2.4.1 Pre-processing features: categorical feature components

We have 4 categorical (nominal) feature components `gas`, `brand`, `area` and `ct` in our data $\mathcal{D}$. These categorical feature components need pre-processing (feature engineering) for the application of the Poisson GLM with regression function (2.2) because they are not real-valued. Component `gas` $\in \{$`Diesel`, `Regular`$\}$ is binary and is transformed to 0 and 1, respectively. The area code is ordinal and is transformed by $\{\mathrm{A}, \ldots, \mathrm{F}\} \mapsto \{1, \ldots, 6\} \subset \mathbb{R}$, see Figure A.9. Thus, there remains the car `brand` and the Swiss cantons `ct`. Car brand has 11 different levels and there are 26 Swiss cantons, see (A.1). We present *dummy coding* to transform these categorical feature components to numerical representations.

For illustrative purposes, we demonstrate dummy coding on the feature component car `brand`. In a first step we use binary coding to illustrate which level a selected car has. In Table 2.1 we provide the one-hot encoding of the car brands on the different rows. Each brand is mapped to a basis vector in $\mathbb{R}^{11}$, i.e. each car brand is represented in one-hot encoding by a unit vector $\boldsymbol{x}^{(1\mathrm{h})} \in \{0, 1\}^{11}$ with $\sum_{l=1}^{11} x_l^{(1\mathrm{h})} = 1$. This mapping is illustrated by the different rows in Table 2.1. In a second step we need to ensure that the resulting design matrix $\mathfrak{X}$ (which includes an intercept component, see (2.7)) has full rank. This is achieved by declaring one level to be the reference level (this level is modeled by the intercept β_0). Dummy coding then only measures relative differences

[2] The true (typically unknown) expected frequency is denoted by $\lambda^\star(\cdot)$ and the corresponding feature space by $\mathcal{X}^\star$, i.e. we have true regression function $\lambda^\star : \mathcal{X}^\star \to \mathbb{R}_+$.

[3] The data is available from `https://people.math.ethz.ch/~wmario/Lecture/MTPL_data.csv`

B1	1	0	0	0	0	0	0	0	0	0	0
B10	0	1	0	0	0	0	0	0	0	0	0
B11	0	0	1	0	0	0	0	0	0	0	0
B12	0	0	0	1	0	0	0	0	0	0	0
B13	0	0	0	0	1	0	0	0	0	0	0
B14	0	0	0	0	0	1	0	0	0	0	0
B2	0	0	0	0	0	0	1	0	0	0	0
B3	0	0	0	0	0	0	0	1	0	0	0
B4	0	0	0	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0	0	1	0
B6	0	0	0	0	0	0	0	0	0	0	1

Table 2.1: One-hot encoding $\boldsymbol{x}^{(1\text{h})} \in \mathbb{R}^{11}$ for car `brand`, encoded on the different rows.

to this reference level. If we declare `B1` to be the reference level, we can drop the first column of Table 2.1. This provides the dummy coding scheme for car `brand` given in Table 2.2.

B1	0	0	0	0	0	0	0	0	0	0
B10	1	0	0	0	0	0	0	0	0	0
B11	0	1	0	0	0	0	0	0	0	0
B12	0	0	1	0	0	0	0	0	0	0
B13	0	0	0	1	0	0	0	0	0	0
B14	0	0	0	0	1	0	0	0	0	0
B2	0	0	0	0	0	1	0	0	0	0
B3	0	0	0	0	0	0	1	0	0	0
B4	0	0	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0	1	0
B6	0	0	0	0	0	0	0	0	0	1

Table 2.2: Dummy coding $\boldsymbol{x}^{\texttt{brand}} \in \mathbb{R}^{10}$ for car `brand`, encoded on the different rows.

We define the part of the feature space $\mathcal{X}$ that belongs to car `brand` as follows

$$\mathcal{X}^{\texttt{brand}} = \left\{ \boldsymbol{x}^{\texttt{brand}} \in \{0,1\}^{10};\ \sum_{l=1}^{10} x_l^{\texttt{brand}} \in \{0,1\} \right\} \ \subset\ \mathbb{R}^{10}. \tag{2.16}$$

That is, the resulting part $\mathcal{X}^{\texttt{brand}}$ of the feature space $\mathcal{X}$ is 10 dimensional, the feature components can only take values 0 and 1, and the components of $\boldsymbol{x}^{\texttt{brand}}$ add up to either 0 or 1, indicating to which particular car `brand` a specific policy is belonging to. Note that this feature space may differ from the original feature space $\mathcal{X}^\star$ of Appendix A. For the 26 Swiss cantons we proceed completely analogously receiving $\mathcal{X}^{\texttt{ct}} \subset \{0,1\}^{25} \subset \mathbb{R}^{25}$ with, for instance, canton `ZH` being the reference level.

Remarks 2.7.

- If we have k categorical classes, then we need $k-1$ indicators (dummy variables) to uniquely identify the parametrization of the (multiplicative) model including an

intercept. Choice (2.16) assumes that one level is the reference level. This reference level is described by the intercept β_0. All other levels are measured relative to this reference level and are described by regression parameters β_l, with $1 \leq l \leq k-1$, see also (2.10). This parametrization is called dummy coding or treatment contrast coding where the reference level serves as control group and all other levels are described by dummy variables relative to this control group.

- Other identification schemes (contrasts) are possible, as long as they lead to a full rank in the design matrix $\mathfrak{X}$. For instance, if we have the hypothesis that the first level is the best, the second level is the second best, etc., then we could consider Helmert's contrast coding diagram given in Table 2.3. For illustrative purposes we only choose $k = 5$ car brands in Table 2.3.

B1	4/5	0	0	0
B2	-1/5	3/4	0	0
B3	-1/5	-1/4	2/3	0
B4	-1/5	-1/4	-1/3	1/2
B5	-1/5	-1/4	-1/3	-1/2

Table 2.3: Helmert's contrast coding $\boldsymbol{x}^{\texttt{Helmert}} \in \mathbb{R}^4$ encoded on the different rows.

This coding in Table 2.3 has the following properties: (i) each column sums to zero, (ii) all columns are orthogonal, and (iii) each level is compared to the mean of the subsequent levels, see also (2.17), below. In view of (2.2), this provides regression function for policy i (restricted to 5 car brand levels, only)

$$\langle \boldsymbol{\beta}, \boldsymbol{x}_i^{\texttt{Helmert}} \rangle = \begin{cases} \beta_0 + \frac{4}{5}\beta_1 & i \text{ is car brand } \texttt{B1}, \\ \beta_0 - \frac{1}{5}\beta_1 + \frac{3}{4}\beta_2 & i \text{ is car brand } \texttt{B2}, \\ \beta_0 - \frac{1}{5}\beta_1 - \frac{1}{4}\beta_2 + \frac{2}{3}\beta_3 & i \text{ is car brand } \texttt{B3}, \\ \beta_0 - \frac{1}{5}\beta_1 - \frac{1}{4}\beta_2 - \frac{1}{3}\beta_3 + \frac{1}{2}\beta_4 & i \text{ is car brand } \texttt{B4}, \\ \beta_0 - \frac{1}{5}\beta_1 - \frac{1}{4}\beta_2 - \frac{1}{3}\beta_3 - \frac{1}{2}\beta_4 & i \text{ is car brand } \texttt{B5}. \end{cases}$$

Observe that the mean over *all* risk classes is described by β_0. For a given level, say car brand B2, the subsequent levels have the same mean (on the log scale) except in the variable to which it is compared to:

$$\begin{aligned} \text{(log-) mean of car brand B2:} \quad & \beta_0 - \frac{1}{5}\beta_1 + \frac{3}{4}\beta_2, \\ \text{(log-) mean over car brands B3, B4, B5:} \quad & \beta_0 - \frac{1}{5}\beta_1 - \frac{1}{4}\beta_2, \end{aligned} \tag{2.17}$$

thus, the difference in this comparison is exactly one unit of β_2.

- The choice of the explicit identification scheme does not have any influence on the prediction, i.e. different (consistent) parametrizations lead to the same prediction. However, the choice of the identification may be important for the interpretation of the parameters (see the examples above) and it is important, in particular, if we explore parameter reduction techniques, e.g., backward selection.

- If we have $k = 2$ categorical classes (binary case), then dummy coding is equivalent to a continuous consideration of that component.

Conclusion 2.8. If we consider the 5 continuous feature components `age`, `ac`, `power`, `area` and log(`dens`) as log-linear, the binary component `gas` as 0 or 1, and the 2 categorical components `brand` and `ct` by dummy coding we receive a feature space $\mathcal{X} = \mathbb{R}^5 \times \{0,1\} \times \mathcal{X}^{\texttt{brand}} \times \mathcal{X}^{\texttt{ct}} \subset \mathbb{R}^q$ of dimension $q = 5 + 1 + 10 + 25 = 41$.

2.4.2 Pre-processing features: continuous feature components

In view of the previous conclusion, we need to ask ourselves whether a log-linear consideration of the continuous feature components `age`, `ac`, `power`, `area` and log(`dens`) in regression function (2.2) is appropriate.

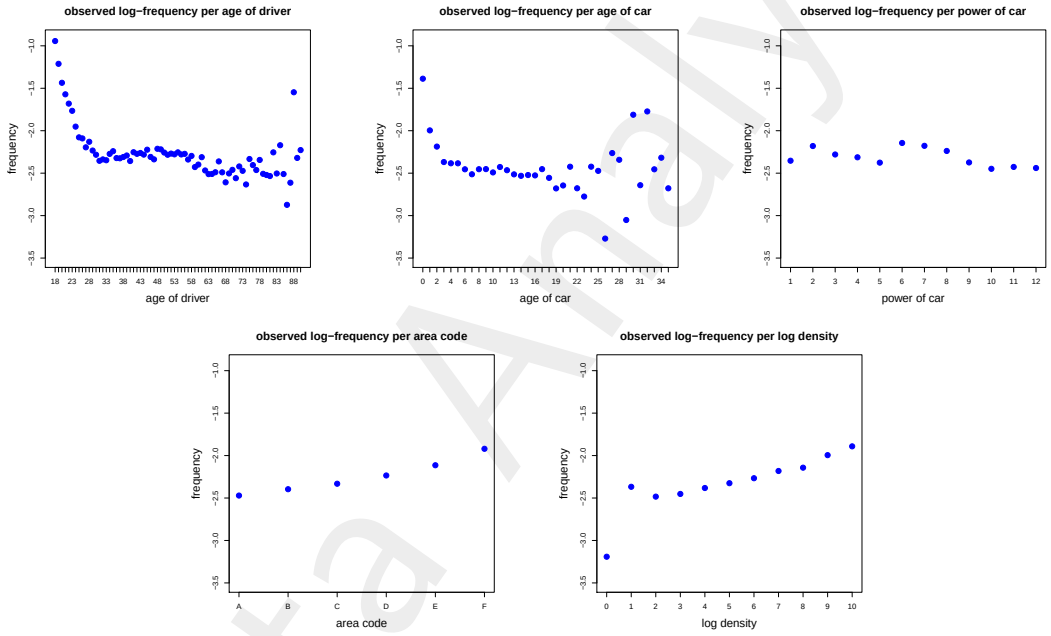


Figure 2.1: Observed marginal log-frequencies of the continuous feature components `age`, `ac`, `power`, `area` and log(`dens`).

In Figure 2.1 we illustrate the observed marginal log-frequencies of the continuous feature components `age`, `ac`, `power`, `area` and log(`dens`) provided by the data $\mathcal{D}$ in Appendix A. We see that some of these graphs are (highly) non-linear and non-monotone which does not support the log-linear assumption in (2.2). This is certainly true for the components `age` and `ac`. The other continuous feature components `power`, `area` and log(`dens`) need a more careful consideration because these marginal plots in Figure 2.1 can be (rather) misleading. Note that these marginal plots involve interactions between feature components and, thus, these marginal plots may heavily be influenced by such interactions. For instance, log(`dens`) at the lower end may suffer from insufficient years at risk and interactions with other feature components that drive low expected frequencies.

GLM modeling decision. To keep the outline of the GLM chapter simple we assume that the components `power`, `area` and log(`dens`) can be modeled by a log-linear approach, and that the components `age` and `ac` need to be modeled differently. These choices are going to be challenged and revised later on.

The first way to deal with non-linear and non-monotone continuous feature components in GLMs is to partition them and then treat them as (nominal) categorical variables. We will present an other treatment in Chapter 3 on generalized additive models (GAMs). In Figure 2.2 we provide the chosen categorization which gives us a partition of `age` into 8 '`age` classes' and of `ac` into 4 '`ac` classes' (which hopefully are homogeneous w.r.t. claims frequency).

`age` class 1:	18-20
`age` class 2:	21-25
`age` class 3:	26-30
`age` class 4:	31-40
`age` class 5:	41-50
`age` class 6:	51-60
`age` class 7:	61-70
`age` class 8:	71-90
`ac` class 0:	0
`ac` class 1:	1
`ac` class 2:	2
`ac` class 3:	3+

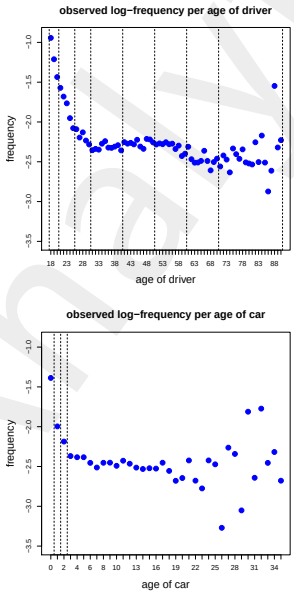


Figure 2.2: (lhs) Chosen '`age` classes' and '`ac` classes'; (rhs) resulting partition.

Listing 2.1: Categorical classes for GLM

```
1 c1 <- c(18, 21, 26, 31, 41, 51, 61, 71, 91)
2 ageGLM <- cbind(c(18:90),c(rep(paste(c1[1],"to",c1[2]-1, sep=""),...
3 dat$ageGLM <- as.factor(ageGLM[dat$age-17,2])
4 dat$ageGLM <- relevel(dat$ageGLM, ref="51to60")
5 dat$acGLM <- as.factor(pmin(dat$ac,3))
6 levels(dat$acGLM) <- c("ac0","ac1","ac2","ac3+")
```

In Listing 2.1 we give the R code to receive this partitioning. Note that we treat these `age` and `ac` classes as categorical feature components. In R this is achieved by declaring these components being of `factor` type, see lines 3 and 5 of Listing 2.1. For the GLM approach we then use dummy coding to bring these (new) categorical feature components into the right structural form, see Section 2.4.1. On line 4 of Listing 2.1 we define

the reference level of 'age class' for this dummy coding, and for 'ac class' we simply choose the first one ac0 as reference level. Thus, in full analogy to Section 2.4.1, we obtain $\mathcal{X}^{\texttt{age}} \subset \{0,1\}^7 \subset \mathbb{R}^7$ for modeling 'age classes', and $\mathcal{X}^{\texttt{ac}} \subset \{0,1\}^3 \subset \mathbb{R}^3$ for modeling 'ac classes'. This implies that our feature space considers 3 log-linear continuous feature components power, area and log(dens), the binary feature component gas and 4 categorical feature components 'age class', 'ac class', brand and ct. This equips us with feature space

$$\mathcal{X} = \mathbb{R}^3 \times \{0,1\} \times \mathcal{X}^{\texttt{age}} \times \mathcal{X}^{\texttt{ac}} \times \mathcal{X}^{\texttt{brand}} \times \mathcal{X}^{\texttt{ct}} \subset \mathbb{R}^q,$$

of dimension $q = 3 + 1 + 7 + 3 + 10 + 25 = 49$.

Remarks 2.9 (feature engineering)**.**

- The choice of 'age classes' and 'ac classes' has been done purely expert based by looking at the marginal plots in Figure 2.2 (rhs). They have been built such that each resulting class is as homogeneous as possible in the underlying frequency. In Chapter 6 we will meet regression trees which allow for a data driven selection of classes.

- Besides the heterogeneity between and within the chosen classes, one should also pay attention to the resulting volumes. The smallest 'age class' 18to20 has a total volume of 1'741.48 years at risk, the smallest 'ac class' ac0 a total volume of 14'009.20 years at risk. The latter is considered to be sufficiently large, the former might be critical. We come back to this in formula (2.20), below. Thus, in building categorical classes one needs to find a good trade-off between homogeneity within the classes and minimal sizes of these classes for reliable parameter estimation.

- A disadvantage of this categorical coding of continuous variables is that the topology gets lost. For instance, after categorical coding as shown in Figure 2.2 it is no longer clear that, e.g., ages 70 and 71 are neighboring ages (in categorical coding). Moreover, the resulting frequency function will not be continuous at the boundaries of the classes. Therefore, alternatively, one could also try to replace non-monotone continuous features by more complex functions. For instance, we could map age to a 4 dimensional function having regression parameters $\beta_l, \ldots, \beta_{l+3} \in \mathbb{R}$

$$\texttt{age} \;\mapsto\; \beta_l \texttt{age} + \beta_{l+1} \log(\texttt{age}) + \beta_{l+2} \texttt{age}^2 + \beta_{l+3} \texttt{age}^3.$$

  In this coding we keep age as a continuous variable and we allow for more modeling flexibility by providing a pre-specified functional form that will run into the regression function.

- In (2.10) we have seen that if we choose the log-link function then we get a multiplicative tariff structure. Thus, all feature components interact in a multiplicative way. If we have indication that interactions have a different structure, we can model this structure explicitly. As an example, we may consider

$$(\texttt{age}, \texttt{ac}) \;\mapsto\; \beta_l \texttt{age} + \beta_{l+1} \texttt{ac} + \beta_{l+2} \texttt{age}/\texttt{ac}^2.$$

This gives us systematic effect (under log-link choice)

$$\exp\{\beta_l \texttt{age}\} \exp\{\beta_{l+1} \texttt{ac}\} \exp\{\beta_{l+2} \texttt{age}/\texttt{ac}^2\}.$$

This shows that GLMs allow for a lot of modeling flexibility, but the modeling has to be done explicitly by the modeler. This requires deep knowledge about the data, so that features can be engineered and integrated in the appropriate way.

Example 2.10 (example GLM1)**.** In a first example we only consider feature information `age` and `ac`, and we choose the (categorical) feature pre-processing as described in Listing 2.1. In the next section on 'data compression' it will become clear why we are starting with this simplified example.
We define feature space $\mathcal{X} = \mathcal{X}^{\texttt{age}} \times \mathcal{X}^{\texttt{ac}} \subset \mathbb{R}^q$ which has dimension $q = 7 + 3 = 10$, and we assume that the regression function $\lambda : \mathcal{X} \to \mathbb{R}_+$ is given by the log-linear form (2.2).

Listing 2.2: Results of example GLM1 (Example 2.10)

```
1  Call:
2  glm(formula = claims ~ ageGLM + acGLM, family = poisson(), data = dat,
3      offset = log(expo))
4
5  Deviance Residuals:
6      Min        1Q    Median        3Q       Max
7  -1.1643   -0.3967   -0.2862   -0.1635    4.3409
8
9  Coefficients:
10               Estimate Std. Error z value Pr(>!z!)
11 (Intercept)   -1.41592    0.02076 -68.220  < 2e-16 ***
12 ageGLM18to20   1.12136    0.04814  23.293  < 2e-16 ***
13 ageGLM21to25   0.48988    0.02909  16.839  < 2e-16 ***
14 ageGLM26to30   0.13315    0.02473   5.383 7.32e-08 ***
15 ageGLM31to40   0.01316    0.01881   0.700  0.48403
16 ageGLM41to50   0.05644    0.01846   3.058  0.00223 **
17 ageGLM61to70  -0.17238    0.02507  -6.875 6.22e-12 ***
18 ageGLM71to90  -0.13196    0.02983  -4.424 9.71e-06 ***
19 acGLMac1      -0.60897    0.02369 -25.708  < 2e-16 ***
20 acGLMac2      -0.79284    0.02588 -30.641  < 2e-16 ***
21 acGLMac3+     -1.08595    0.01866 -58.186  < 2e-16 ***
22 ---
23 Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
24
25 (Dispersion parameter for poisson family taken to be 1)
26
27     Null deviance: 145532  on 499999  degrees of freedom
28 Residual deviance: 141755  on 499989  degrees of freedom
29 AIC: 192154
30
31 Number of Fisher Scoring iterations: 6
```

The MLE $\widehat{\boldsymbol{\beta}} \in \mathbb{R}^{q+1}$ is then calculated using the R package `glm`. The results are presented in Listing 2.2. We receive $q + 1 = 11$ estimated MLE parameters (lines 11-21), the intercept being $\widehat{\beta}_0 = -1.41592$. Thus, the reference risk cell $(\texttt{51to60}, \texttt{ac0})$ has an expected frequency of $\widehat{\lambda}(\boldsymbol{x}) = \exp\{-1.41592\} = 24.27\%$. All other risk cells are measured relative to this reference cell using the multiplicative structure (2.10).

The further columns on lines 11-21 provide: column `Std. Error` gives the estimated standard deviation $\widehat{\sigma}_l$ in $\widehat{\beta}_l$, for details we refer to (2.33) in the appendix Section 2.6 where Fisher's information matrix is discussed; the column `z value` provides the rooted Wald statistics under the null hypothesis $\beta_l = 0$ (for each component $0 \leq l \leq q$ individually), it is defined by $z_l = \widehat{\beta}_l / \widehat{\sigma}_l$; finally, the last column `Pr(>!z!)` gives the resulting (individual two-sided) p-values for these null hypotheses under asymptotic normality. These null hypotheses should be interpreted as there does not exist a significant difference between the considered level $1 \leq l \leq q$ and the reference level (when using dummy coding). Note that these tests cannot be used to decide whether a categorical variable should be included in the model or not.[4]

Line 28 of Listing 2.2 gives the unscaled in-sample Poisson deviance loss, i.e. $n\mathcal{L}_D^{\text{is}} = D^*(\boldsymbol{N}, \widehat{\lambda}) = 141'755$. This results in an in-sample loss of $\mathcal{L}_D^{\text{is}} = 28.3510 \cdot 10^{-2}$, see also Table 2.4. Line 27 gives the corresponding value of the homogeneous model (called null model) only considering an intercept, this is the model of Section 1.4. This allows us to consider the likelihood ratio test statistics (2.15) for the null hypothesis H_0 of the homogeneous model versus the alternative heterogeneous model considered in this example.

$$\chi^2_{\mathcal{D}} = D^*(\boldsymbol{N}, \widehat{\lambda}_{H_0}) - D^*(\boldsymbol{N}, \widehat{\lambda}_{\text{full}}) = 145'532 - 141'755 = 3'777.$$

This test statistics has approximately a χ^2-distribution with $q = 10$ degrees of freedom. The resulting p-value is almost zero which means that we highly reject the homogeneous model in favor of the model in this example.

Finally, Akaike's information criterion (AIC) is given, and Fisher's scoring method for parameter estimation has converged in 6 iterations.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.1) GLM1	3.1s	11	28.3543	28.3544	0.6052	28.3510	10.2691%

Table 2.4: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 1.1.

Exactly in the same structure as in Table 1.1, we present the corresponding cross-validation losses $\mathcal{L}_D^{\text{CV}}$, the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ and the in-sample loss $\mathcal{L}_D^{\text{is}}$ on line (Ch2.1) of Table 2.4. Note that the estimation loss is given by

[4] The rooted Wald statistics uses a second order Taylor approximation to the log-likelihood. Based on asymptotic normality it allows for a z-test under known dispersion or a t-test for estimated dispersion whether a variable can be dropped from the full model. However, if we have categorical features, one has to be more careful because in this case the p-value only indicates whether a level significantly differs from the reference level.

$$\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star) = \frac{1}{n} \sum_{i=1}^{n} 2v_i \left[\widehat{\lambda}(\boldsymbol{x}_i) - \lambda^\star(\boldsymbol{x}_i) - \lambda^\star(\boldsymbol{x}_i) \log \left(\frac{\widehat{\lambda}(\boldsymbol{x}_i)}{\lambda^\star(\boldsymbol{x}_i)} \right) \right] = 0.6052 \cdot 10^{-2},$$

and this estimation loss can only be calculated because we know the true model $\lambda^\star$, here. We observe that these loss figures are clearly better than in the homogeneous model on line (Ch1.1), but worse compared to the true model $\lambda^\star$ on line (ChA.1). We also observe that we have a negligible over-fitting because cross-validation provides almost the same numbers compared to the in-sample loss.[5]

Next we estimate the dispersion parameter ϕ. The resulting Pearson's and deviance estimators are

$$\widehat{\phi}_P = 1.0061 \qquad \text{and} \qquad \widehat{\phi}_D = 28.3516 \cdot 10^{-2}.$$

The Pearson's estimator shows a small over-dispersion, the deviance estimator is more difficult to interpret because its true level for small frequencies is typically not exactly known, see also Figure 1.1. Note that the deviance dispersion estimate (2.14) scales with the number of parameters q, that is, the degrees of freedom are given by $500'000-(q+1) = 499'989$. This slightly corrects for model complexity. However, the resulting estimate is bigger than the cross-validation loss in our example.

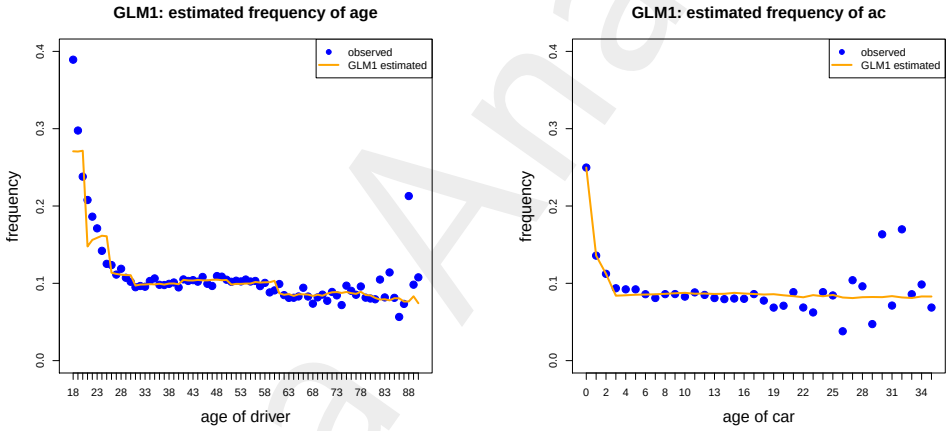


Figure 2.3: Estimated marginal frequencies in example GLM1 for `age` and `ac`.

In Figure 2.3 we present the resulting marginal frequency estimates (averaged over our portfolio); these are compared to the marginal observations. The orange graphs reflect the MLEs provided in Listing 2.2. The graphs are slightly wiggly which is caused by the fact that we have a heterogeneous portfolio that implies non-homogeneous multiplicative interactions between the feature components `age` and `ac`.

Finally, in Figure 2.4 we show the resulting Pearson's and deviance residuals. The different colors illustrate the underlying years at risk $v_i \in (0, 1]$, $i = 1, \ldots, n$. As previously mentioned, we observe that Pearson's residuals are not very robust for small years at risk v_i (red color) because we divide by these (small) volume measures for Pearson's residual

[5] Cross-validation is done on the same partition for all models considered in these notes. Run time measures the time to fit the model once on a personal laptop Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz with 16GB RAM.

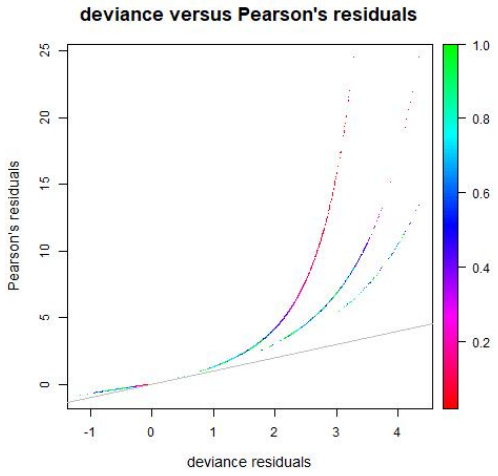


Figure 2.4: Deviance residuals $\widehat{\delta}_i^D$ versus Pearson's residuals $\widehat{\delta}_i^P$; the colors illustrate the underlying years at risk $v_i \in (0, 1]$.

definition (2.12). In general, one should work with deviance residuals because this is a distribution-adapted version of the residuals and considers skewness and tails in the right way (supposed that the model choice is appropriate). This finishes the example for the time being. ■

2.4.3 Data compression

We note that Example 2.10 only involves $d = 8 \cdot 4 = 32$ different risk cells for the corresponding 'age classes' and 'ac classes', see also Figure 2.2. Within these risk cells we assume that the individual insurance policies are homogeneous, i.e. can be characterized by a common feature value $\boldsymbol{x}_k^+ \in \mathcal{X} = \mathcal{X}^{\texttt{age}} \times \mathcal{X}^{\texttt{ac}} \subset \mathbb{R}^q$ with $q = 10$. Running the R code in Listing 2.2 may be time consuming if the number of policies n is large. Therefore, we could/should try to first compress the data accordingly. For independent Poisson distributed random variables this is rather simple, the aggregation property of Lemma 1.3 implies that we can consider sufficient statistics on the $d = |\mathcal{X}| = 32$ different risk cells. These are described by the (representative) features $\{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_d^+\} = \mathcal{X}$. We define the aggregated portfolios in each risk cell $k = 1, \ldots, d$ by

$$N_k^+ = \sum_{i=1}^n N_i \mathbb{1}_{\{\boldsymbol{x}_i = \boldsymbol{x}_k^+\}} \qquad \text{and} \qquad v_k^+ = \sum_{i=1}^n v_i \mathbb{1}_{\{\boldsymbol{x}_i = \boldsymbol{x}_k^+\}}. \tag{2.18}$$

Observe that N_i and v_i are on an individual insurance policy level, whereas N_k^+ and v_k^+ are on an aggregated risk cell (portfolio) level. The consideration of the latter provides a substantial reduction in computational time when calculating the MLE $\widehat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ because the $n = 500'000$ observations are compressed to $d = 32$ aggregated observations (sufficient statistics). Lemma 1.3 implies that all risk cells $k = 1, \ldots, d$ are independent and Poisson distributed

$$N_k^+ \sim \text{Poi}\left(\lambda(\boldsymbol{x}_k^+) v_k^+\right).$$

The joint log-likelihood function on the compressed data $\mathcal{D}^+ = \{(N_k^+, \boldsymbol{x}_k^+, v_k^+);\ k = 1, \ldots, d\}$ is given by

$$\ell_{\boldsymbol{N}^+}(\boldsymbol{\beta}) = \sum_{k=1}^{d} -\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_k^+ \rangle v_k^+ + N_k^+ \left(\langle \boldsymbol{\beta}, \boldsymbol{x}_k^+ \rangle + \log v_k^+ \right) - \log(N_k^+!). \tag{2.19}$$

We introduce the design matrix $\mathfrak{X}^+ = (x_{k,l}^+)_{1 \le k \le d, 0 \le l \le q}$ as above, and the MLE $\widehat{\boldsymbol{\beta}}$ is found as in Proposition 2.2.

Listing 2.3: Aggregation/compression of risk cells

```
> dat <- ddply(dat, .(ageGLM, acGLM), summarize,expo=sum(expo),claims=sum(claims))
> str(dat)
'data.frame':   32 obs. of  4 variables:
 $ ageGLM: Factor w/ 8 levels "51to60","18to20",..: 1 1 1 1 2 2 2 2 3 3 ...
 $ acGLM : Factor w/ 4 levels "ac0","ac1","ac2",..: 1 2 3 4 1 2 3 4 1 2 ...
 $ expo  : num  3144.4 6286.4 5438.8 38209.8 38.6 ...
 $ claims: int  712 817 618 3174 19 31 12 410 179 192 ...
```

In Listing 2.3 we provide the R code for the data compression in each risk cell. Note that `ageGLM` and `acGLM` describe the categorical classes, see also Listing 2.1.

Example 2.10, revisited (example GLM2)**.** We revisit example GLM1 (Example 2.10), but calculate the MLE $\widehat{\boldsymbol{\beta}}$ directly on the aggregated risk cells received from Listing 2.3. The results are presented in Listing 2.4.

We note that the result for the MLE $\widehat{\boldsymbol{\beta}}$ is exactly the same, compare Listings 2.2 and 2.4. Of course, this needs to be the case due to the fact that we work with (aggregated) sufficient statistics in the latter version. The only things that change are the deviance losses because we work on a different scale now. We receive in-sample loss $\mathcal{L}_{D^+}^{\text{is}} = D^*(\boldsymbol{N}^+, \widehat{\lambda})/d = 32.1610/32 = 1.0050$ on the aggregate data $\mathcal{D}^+$. The degrees of freedom are $d - (q+1) = 32 - 11 = 21$, this results in dispersion parameter estimates

$$\widehat{\phi}_P = 1.5048 \qquad \text{and} \qquad \widehat{\phi}_D = 1.5315.$$

Thus, we obtain quite some over-dispersion which indicates that our regression function choice misses important structure.

In Figure 2.5 (lhs) we plot deviance residuals versus Pearson's residuals, the different colors showing the volumes of the underlying risk cells. We note that the two versions of residuals are almost identical on an aggregate risk cell level. Figure 2.5 (rhs) gives the Tukey–Anscombe plot which plots the residuals versus the fitted means. In this plot we would not like to discover any structure.

In Figure 2.6 we plot the estimated marginal frequencies of example GLM2 (on the different categorical levels). We notice that the observed frequencies N_k^+/v_k^+ (marginally aggregated) exactly match the estimated expected frequencies $\widehat{\lambda}(\boldsymbol{x}_k^+)$ (also marginally aggregated). This is explained by the fact that the Poisson GLM method with categorical coding provides identical results to the method of the total marginal sums by Bailey [7] and Jung [76], see Section 7.1 in Wüthrich [135]. This also explains why the average estimated frequencies of the homogeneous model and example GLM1 are identical in the last column of Table 2.4. This finishes example GLM2. ■

Listing 2.4: Results of example GLM2

```
Call:
glm(formula = claims ~ ageGLM + acGLM, family = poisson(), data = dat,
    offset = log(expo))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9447  -0.8175  -0.1509   0.6370   1.9360

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.41592    0.02076 -68.220  < 2e-16 ***
ageGLM18to20  1.12136    0.04814  23.293  < 2e-16 ***
ageGLM21to25  0.48988    0.02909  16.839  < 2e-16 ***
ageGLM26to30  0.13315    0.02473   5.383 7.32e-08 ***
ageGLM31to40  0.01316    0.01881   0.700  0.48403
ageGLM41to50  0.05644    0.01846   3.058  0.00223 **
ageGLM61to70 -0.17238    0.02507  -6.875 6.22e-12 ***
ageGLM71to90 -0.13196    0.02983  -4.424 9.71e-06 ***
acGLMac1     -0.60897    0.02369 -25.708  < 2e-16 ***
acGLMac2     -0.79284    0.02588 -30.641  < 2e-16 ***
acGLMac3+    -1.08595    0.01866 -58.186  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 3809.473  on 31  degrees of freedom
Residual deviance:   32.161  on 21  degrees of freedom
AIC: 305.12

Number of Fisher Scoring iterations: 3
```

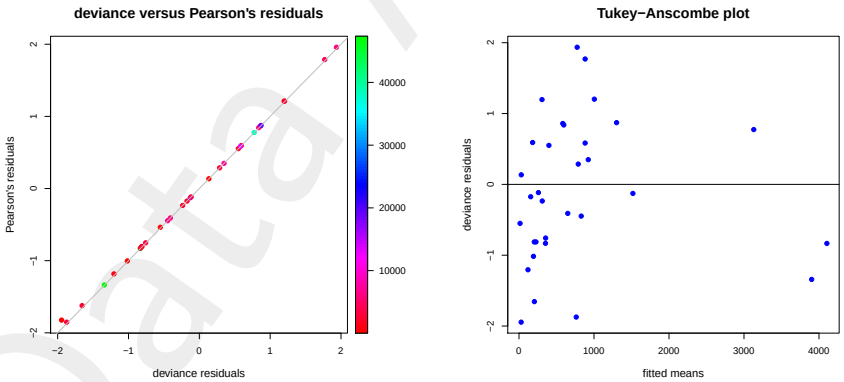


Figure 2.5: Example GLM2: (lhs) deviance versus Pearson's residuals, (rhs) Tukey–Anscombe plot.

2.4.4 Issue about low frequencies

In the previous section we have worked on aggregated data $\mathcal{D}^+$ (sufficient statistics). This aggregation is possible on categorical classes, but not necessarily if we consider continuous feature components. This is exactly the reason why we have been starting

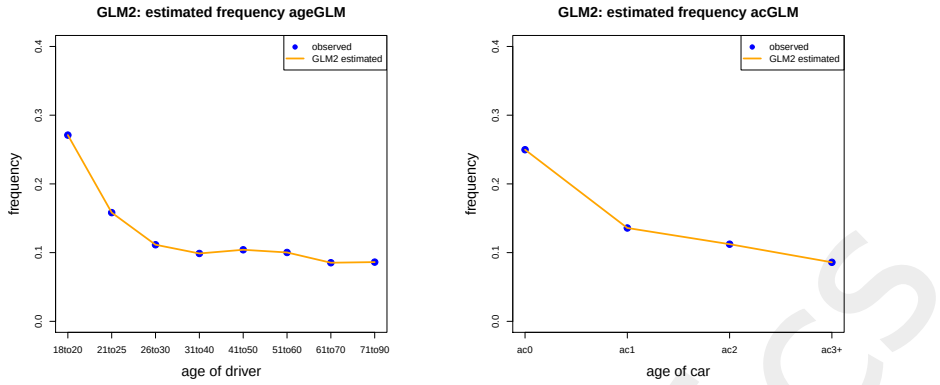


Figure 2.6: Estimated marginal frequencies in GLM2 Example 2.10 for 'age classes' and 'ac classes' risk cells.

by only considering two categorized feature components in Example 2.10 (GLM1 and GLM2). In this section we would like to emphasize a specific issue in insurance of having rather low expected frequencies in the range of 3% to 20%. If, for instance, $\lambda(\boldsymbol{x}) = 5\%$ and $v = 1$ then we obtain for $N \sim \text{Poi}(\lambda(\boldsymbol{x})v)$

$$\mathbb{E}[N] = 0.05 \qquad \text{and} \qquad \text{Var}(N)^{1/2} = 0.22.$$

This indicates that the pure randomness is typically of much bigger magnitude than possible structural differences (see also Figure 1.2), and we require a lot of information to distinguish good from bad car drivers. In particular, we need portfolios having appropriate volumes (years at risk). If, instead, these drivers would have observations of 10 years at risk, i.e. $v = 10$, then magnitudes of order start to change

$$\mathbb{E}[N] = 0.50 \qquad \text{and} \qquad \text{Var}(N)^{1/2} = 0.71.$$

If we take confidence bounds of 2 standard deviations we obtain for the expected frequency $\lambda(\boldsymbol{x})$ the following interval

$$\left[\lambda(\boldsymbol{x}) - 2\sqrt{\frac{\lambda(\boldsymbol{x})}{v}},\ \lambda(\boldsymbol{x}) + 2\sqrt{\frac{\lambda(\boldsymbol{x})}{v}}\right]. \tag{2.20}$$

This implies for $\lambda(\boldsymbol{x}) = 5\%$ that we need $v = 2'000$ years at risk to detect a structural difference to an expected frequency of 4%. Of course, this is taken care of by building sufficiently large homogeneous sub-portfolios (we have $n = 500'000$ policies). But if the dimension of the feature space $\mathcal{X}$ is too large or if we have too many categorical feature components then we cannot always achieve to obtain sufficient volumes for parameter estimation (and a reduction of dimension technique may need to be applied). Note that the smallest 'age class' `18to20` is rather heterogeneous but it only has a total volume of 1'741.48 years at risk, see Remarks 2.9. Moreover, these considerations also refer to the remark after Example 2.1.

2.4.5 Models GLM3+ considering all feature components

In this section we consider the a GLM considering all feature components and based on our GLM modeling decision on page 40: we assume that the components `power`, `area` and log(`dens`) can be modeled by a log-linear approach, component `gas` is binary, the components `age` and `ac` are modeled categorically according to Figure 2.2, and the remaining feature components `brand` and `ct` are categorical. This gives us the feature space

$$\mathcal{X} = \mathbb{R}^3 \times \{0,1\} \times \mathcal{X}^{\texttt{age}} \times \mathcal{X}^{\texttt{ac}} \times \mathcal{X}^{\texttt{brand}} \times \mathcal{X}^{\texttt{ct}} \subset \mathbb{R}^q,$$

of dimension $q = 3 + 1 + 7 + 3 + 10 + 25 = 49$. We call this model example GLM3.

Listing 2.5: Results of example GLM3

```
Call:
glm(formula = claims ~ power + area + log(dens) + gas + ageGLM +
    acGLM + brand + ct, family = poisson(), data = dat, offset = log(expo))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1944  -0.3841  -0.2853  -0.1635   4.3759

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.7374212  0.0455594 -38.135  < 2e-16 ***
power        -0.0008945  0.0031593  -0.283 0.777069
area          0.0442176  0.0192701   2.295 0.021755 *
log(dens)     0.0318978  0.0143172   2.228 0.025884 *
gasRegular    0.0491739  0.0128676   3.822 0.000133 ***
ageGLM18to20  1.1409863  0.0483603  23.593  < 2e-16 ***
.
ageGLM71to90 -0.1264737  0.0300539  -4.208 2.57e-05 ***
acGLMac1     -0.6026905  0.0237064 -25.423  < 2e-16 ***
acGLMac2     -0.7740611  0.0259994 -29.772  < 2e-16 ***
acGLMac3+    -1.0242291  0.0204606 -50.059  < 2e-16 ***
brandB10     -0.0058938  0.0445983  -0.132 0.894864
.
brandB5       0.1300453  0.0292474   4.446 8.73e-06 ***
brandB6      -0.0004378  0.0332226  -0.013 0.989486
ctAG         -0.0965560  0.0274785  -3.514 0.000442 ***
.
ctVS         -0.1110315  0.0317446  -3.498 0.000469 ***
ctZG         -0.0724881  0.0463526  -1.564 0.117855
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

    Null deviance: 145532  on 499999  degrees of freedom
Residual deviance: 140969  on 499950  degrees of freedom
AIC: 191445
```

The results for the MLE $\widehat{\boldsymbol{\beta}}$ are provided in Listing 2.5 and the run time is found in Table 2.5. From Listing 2.5 we need to question the modeling of (all) continuous variables `power`, `area` and log(`dens`). Either these variables should not be in the model or we should consider them in a different functional form.

In Table 2.5 we state the resulting loss figures of model GLM3. They are better than the ones of model GLM1 (only considering the two `age` and `ac` classes, respectively), but there is still room for improvements compared to the true model $\lambda^\star$.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.1) GLM1	3.1s	11	28.3543	28.3544	0.6052	28.3510	10.2691%
(Ch2.3) GLM3	12.0s	50	28.2125	28.2133	0.4794	28.1937	10.2691%
(Ch2.4) GLM4	14.0s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch2.5) GLM5	13.3s	56	28.1508	28.1520	0.4128	28.1292	10.2691%

Table 2.5: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 2.4.

As a first model modification we consider `power` as categorical, merging powers above and including 9 to one class, thus, we consider 9 different '`power` classes'. This equips us with a new feature space

$$\mathcal{X} = \mathcal{X}^{\texttt{power}} \times \mathbb{R}^2 \times \{0,1\} \times \mathcal{X}^{\texttt{age}} \times \mathcal{X}^{\texttt{ac}} \times \mathcal{X}^{\texttt{brand}} \times \mathcal{X}^{\texttt{ct}} \subset \mathbb{R}^q, \tag{2.21}$$

of dimension $q = 8 + 2 + 1 + 7 + 3 + 10 + 25 = 56$. We call this new model GLM4. The results are presented in Listing 2.6.

Based on this analysis we keep all components in the model. In particular, we should keep the variable `power` but a log-linear shape is not the right functional form to consider `power`. This also becomes apparent from the cross-validation analysis given in Table 2.5 on line (Ch2.4). Note that the number of parameters has been increasing from 11 (GLM1) to 57 (GLM4), and at the same time over-fitting is increasing (difference between cross-validation loss $\mathcal{L}_D^{\text{CV}}$ and in-sample loss $\mathcal{L}_D^{\text{is}}$). The cross-validation loss has decreased by $28.3543 - 28.1502 = 0.2041$ which is consistent with the decrease of 0.1915 in estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ from GLM1 to GLM4.

Another important note is that all GLMs fulfill the balance property of Proposition 2.4. This is also seen from the last column in Table 2.5.

Next we consider an analysis of variance (ANOVA) which nests GLMs w.r.t. the considered feature components. In this ANOVA, terms are sequentially added to the model. Since the order of this sequentially adding is important, we re-order the components on lines 2 and 3 of Listing 2.6 as follows `acGLM`, `ageGLM`, `ct`, `brand`, `powerGLM`, `gas`, log(`dens`) and `area`. This ordering is based on the rationale that the first one is the most important feature component and the last one is the least important one. The ANOVA is then done in R by the command `anova`.

In Listing 2.7 we present the results of this ANOVA. The column `Df` shows the number of model parameters β_l, $1 \le l \le q$, the corresponding feature component uses. The column `Deviance` shows the amount of reduction in in-sample deviance loss $D^*(\boldsymbol{N}, \widehat{\lambda})$ when sequentially adding this feature component, and the last column `Resid.Dev` shows the remaining in-sample deviance loss. We note that `gas` and `area` lead to a comparably small decrease of in-sample loss which may question the use of these feature components (in the current form). We can calculate the corresponding p-values of the χ^2-test statistics

Listing 2.6: Results of example GLM4

```
Call:
glm(formula = claims ~ powerGLM + area + log(dens) + gas + ageGLM +
    acGLM + brand + ct, family = poisson(), data = dat, offset = log(expo))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1373  -0.3820  -0.2838  -0.1624   4.3856

Coefficients:
               Estimate Std. Error z value Pr(>!z!)
(Intercept)  -1.903e+00  4.699e-02 -40.509  < 2e-16 ***
powerGLM2     2.681e-01  2.121e-02  12.637  < 2e-16 ***
powerGLM3     2.525e-01  2.135e-02  11.828  < 2e-16 ***
powerGLM4     1.377e-01  2.113e-02   6.516 7.22e-11 ***
powerGLM5    -2.498e-02  3.063e-02  -0.816 0.414747
powerGLM6     3.009e-01  3.234e-02   9.304  < 2e-16 ***
powerGLM7     2.214e-01  3.240e-02   6.835 8.22e-12 ***
powerGLM8     1.103e-01  4.128e-02   2.672 0.007533 **
powerGLM9    -1.044e-01  4.708e-02  -2.218 0.026564 *
area          4.333e-02  1.927e-02   2.248 0.024561 *
log(dens)     3.224e-02  1.432e-02   2.251 0.024385 *
gasRegular    6.868e-02  1.339e-02   5.129 2.92e-07 ***
ageGLM18to20  1.142e+00  4.836e-02  23.620  < 2e-16 ***
.
.
ctZG         -8.123e-02  4.638e-02  -1.751 0.079900 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 145532  on 499999  degrees of freedom
Residual deviance: 140641  on 499943  degrees of freedom
AIC: 191132
```

Listing 2.7: ANOVA results 1

```
Analysis of Deviance Table

Model: poisson, link: log

Response: claims

Terms added sequentially (first to last)


          Df Deviance Resid. Df Resid. Dev
NULL                    499999     145532
acGLM      3  2927.32     499996     142605
ageGLM     7   850.00     499989     141755
ct        25   363.29     499964     141392
brand     10   124.37     499954     141267
powerGLM   8   315.48     499946     140952
gas        1    50.53     499945     140901
log(dens)  1   255.22     499944     140646
area       1     5.06     499943     140641
```

Listing 2.8: ANOVA results 2

```
Analysis of Deviance Table

Model: poisson, link: log

Response: claims

Terms added sequentially (first to last)


           Df Deviance Resid. Df Resid. Dev
NULL                      499999     145532
acGLM       3  2927.32    499996     142605
ageGLM      7   850.00    499989     141755
ct         25   363.29    499964     141392
brand      10   124.37    499954     141267
powerGLM    8   315.48    499946     140952
gas         1    50.53    499945     140901
area        1   255.20    499944     140646
log(dens)   1     5.07    499943     140641
```

Listing 2.9: drop 1 analysis

```
Single term deletions

Model:
claims ~ acGLM + ageGLM + ct + brand + powerGLM + gas + areaGLM +
    log(dens)
           Df Deviance    AIC     LRT  Pr(>Chi)
<none>          140641 191132
acGLM       3   142942 193426 2300.61 < 2.2e-16 ***
ageGLM      7   141485 191962  843.91 < 2.2e-16 ***
ct         25   140966 191406  324.86 < 2.2e-16 ***
brand      10   140791 191261  149.70 < 2.2e-16 ***
powerGLM    8   140969 191443  327.68 < 2.2e-16 ***
gas         1   140667 191156   26.32 2.891e-07 ***
areaGLM     1   140646 191135    5.06   0.02453 *
log(dens)   1   140646 191135    5.07   0.02434 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

(2.15) with `Df` degrees of freedom. These p-values are all close to zero except for `area` and `log(dens)`. These p-value are 2.5%, which may allow to work with a model reduced by `area` and/or `log(dens)`.

One may argue that the ANOVA in Listing 2.7 is not "fully fair" for `area` because this feature component is considered as the last one in this sequential analysis. Therefore, we revert the order in a second analysis putting `log(dens)` to the end of the list, see Listing 2.8. Indeed, in this case we come to the conclusion that feature component `log(dens)` may not be necessary. In fact, these two feature components seem to be exchangeable which is explained by the fact that they are very highly correlated, see Table A.2 in the appendix.

For these reasons we study another model GLM5 where we completely drop `area`. The cross-validation results are provided on line (Ch2.5) of Table 2.5. These cross-validation

losses $\mathcal{L}_D^{\text{CV}}$ are slightly bigger for model GLM5 compared to model GLM4, therefore, we decide to keep `area` in the model, and we work with feature space (2.21) for the GLM. Remark that this decision is wrong because the estimation loss of 0.4128 in model GLM5 is smaller than the 0.4137 of model GLM4. However, these estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ are not available in typical applications where the true data generating mechanism $\lambda^\star$ is not known.

	AIC
(Ch2.3) GLM3	191'445
(Ch2.4) GLM4	191'132
(Ch2.5) GLM5	191'135

Table 2.6: Akaike's information criterion AIC.

Alternatively to cross-validation, we could also compare Akaike's information criterion (AIC). In general, the model with the smallest AIC value should be preferred; for the validity of the use of AIC we refer to Section 4.2.3 in [141]. In view of Table 2.6 we reach to the same conclusion as with cross-validation losses in this example.
The ANOVA in Listing 2.7 adds one feature component after the other. In practice, one usually does model selection rather by backward selection. Therefore, one starts with a complex/full model and drops recursively the least significant variables. If we start with the full model we can perform a `drop1` analysis which is given in Listing 2.9: this analysis individually drops each variable from the full model. Based on this analysis we would first drop `areaGLM` because it has the highest p-value (received from the Wald statistics, see Section 2.6). However, this p-value is smaller than 5%, therefore, we would not drop any variable on a 5% significance level. The same conclusion is drawn from AIC because the full model has the smallest value. This finishes the GLM example. ■

2.4.6 Generalized linear models: summary

In the subsequent chapters we will introduce other regression model approaches. These approaches will challenge our (first) model choice GLM4 having $q + 1 = 57$ parameters and feature space (2.21). We emphasize the following points:

- We did not fully fine-tune our model choice GLM4. That is, having $n = 500'000$ observations we could provide better feature engineering. For instance, we may explore explicit functional forms for `ac` or `age`. Moreover, in feature engineering we should also explore potential interactions between the feature components, see Remarks 2.9.

- Exploring functional forms, for instance, for `age` also has the advantage that neighboring '`age` classes' stand in a neighborhood relationship to each other. This is not the case with the categorization used in Figure 2.2.

- Another issue we are going to meet below is over-parametrization and over-fitting. To prevent from over-fitting one may apply regularization techniques which may tell us that certain parameters or levels/labels are not needed. Regularization will be discussed in Section 4.3.2, below.

- Finally, we may also question the Poisson model assumption. In many real applications one observes so-called zero-inflated claims counts which means that there are too many zero observations in the data. In this case often a zero-inflated Poisson (ZIP) model is used that adds an extra point mass to zero. If we face over-dispersion, then also an negative binomial model should be considered.

2.5 Classification problem

For classification we consider the following data

$$\mathcal{D} = \{(Y_1, \boldsymbol{x}_1), \ldots, (Y_n, \boldsymbol{x}_n)\}, \tag{2.22}$$

with features $\boldsymbol{x}_i \in \mathcal{X}$ and responses Y_i taking values in a finite set $\mathcal{Y}$. For instance, we may consider genders $\mathcal{Y} = \{\text{female}, \text{male}\}$. In general, we call the elements of $\mathcal{Y}$ classes or levels, and we represent the classes by a finite set of integers (labels) $\mathcal{Y} = \{0, \ldots, J-1\}$, for a given $J \in \mathbb{N}$. These classes can either be of ordered type (e.g. small, middle, large) or of categorical type (e.g. female, male). Our goal is to construct a classification on $\mathcal{X}$. That is, we aim at constructing a *classifier*

$$\mathcal{C} : \mathcal{X} \to \mathcal{Y}, \qquad \boldsymbol{x} \mapsto y = \mathcal{C}(\boldsymbol{x}). \tag{2.23}$$

This classifier may, for instance, describe the most likely outcome $y \in \mathcal{Y}$ of a (noisy) response Y having feature $\boldsymbol{x}$. The classifier $\mathcal{C}$ provides a finite partition of the feature space $\mathcal{X}$ given by

$$\mathcal{X} = \bigcup_{y \in \mathcal{Y}} \mathcal{X}(y), \qquad \mathcal{X}(y) = \{\boldsymbol{x} \in \mathcal{X};\ \mathcal{C}(\boldsymbol{x}) = y\}. \tag{2.24}$$

2.5.1 Classification of random binary outcomes

Typically, also in classification the data generating mechanism is not known. Therefore, we make a model assumption under which we infer a classifier $\mathcal{C}$ of the given data $\mathcal{D}$. Let us focus on the *binary* situation $\mathcal{Y} = \{0, 1\}$. We assume that all responses of the cases $(Y_i, \boldsymbol{x}_i)$ in $\mathcal{D}$ are independent and generated by the following probability law

$$\pi_1(\boldsymbol{x}) = \mathbb{P}[Y = 1] = p(\boldsymbol{x}) \qquad \text{and} \qquad \pi_0(\boldsymbol{x}) = \mathbb{P}[Y = 0] = 1 - p(\boldsymbol{x}), \tag{2.25}$$

for a given (but unknown) probability function

$$p : \mathcal{X} \to [0, 1], \qquad \boldsymbol{x} \mapsto p(\boldsymbol{x}). \tag{2.26}$$

Formula (2.25) describes a Bernoulli random variable Y. A classifier $\mathcal{C} : \mathcal{X} \to \{0, 1\}$ can then be defined by

$$\mathcal{C}(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ \pi_y(\boldsymbol{x}),$$

with a deterministic rule if both $\pi_y(\boldsymbol{x})$ are equally large. $\mathcal{C}(\boldsymbol{x})$ is the most likely outcome of $Y = Y(\boldsymbol{x})$. Of course, this can be generalized to multiple response classes $J \geq 2$ with corresponding probabilities $\pi_0(\boldsymbol{x}), \ldots, \pi_{J-1}(\boldsymbol{x})$ for features $\boldsymbol{x} \in \mathcal{X}$. This latter

distribution is called categorical distribution. For simplicity we restrict to the binary (Bernoulli) case here.
If for all features $\boldsymbol{x}$ there exists $y \in \mathcal{Y}$ with $\pi_y(\boldsymbol{x}) = 1$, there is no randomness involved, i.e. the responses Y are not noisy, and we obtain a deterministic classification problem. Nevertheless, also in this case we typically need to infer the unknown partition (2.24).

2.5.2 Logistic regression classification

Assume we consider the binary situation and the probability function (2.26) can be described by a logistic functional form through the scalar product $\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle$ given in (2.2). This means that we assume for given $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$

$$p(\boldsymbol{x}) = \frac{\exp\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle}{1 + \exp\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle}, \qquad \text{or equivalently} \qquad \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = \log\left(\frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}\right). \tag{2.27}$$

The aim is to estimate $\boldsymbol{\beta}$ with MLE methods. Assume we have n independent responses in $\mathcal{D}$, given by (2.22), all being generated by a model of the form (2.25)-(2.26). The joint log-likelihood under assumption (2.27) is then given by, we set $\boldsymbol{Y} = (Y_1, \ldots, Y_n)'$,

$$\ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} Y_i \log p(\boldsymbol{x}_i) + (1 - Y_i) \log(1 - p(\boldsymbol{x}_i)) = \sum_{i=1}^{n} Y_i \, \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle - \log(1 + \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle).$$

We calculate the partial derivatives of the log-likelihood function for $0 \leq l \leq q$

$$\frac{\partial}{\partial \beta_l} \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) \;\; = \;\; \sum_{i=1}^{n} \left(Y_i - \frac{\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle}{1 + \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle} \right) x_{i,l} \;\; = \;\; 0.$$

Proposition 2.11. *The solution $\widehat{\boldsymbol{\beta}}$ to the MLE problem* (2.25)-(2.26) *in the logistic case is given by the solution of*

$$\mathfrak{X}' \frac{\exp\{\mathfrak{X}\boldsymbol{\beta}\}}{1 + \exp\{\mathfrak{X}\boldsymbol{\beta}\}} = \mathfrak{X}'\boldsymbol{Y},$$

for design matrix $\mathfrak{X}$ given by (2.7) *and where the ratio is understood element-wise.*

If the design matrix $\mathfrak{X}$ has full rank $q+1 \leq n$, the log-likelihood function is concave and, henceforth, the MLE is unique. Moreover, the root search problem of the score function in Proposition 2.11 is solved by Fisher's scoring method or the IRLS algorithm, see also Section 2.2.

The estimated logistic probabilities are obtained by

$$\widehat{\pi}_1(\boldsymbol{x}) = \widehat{p}(\boldsymbol{x}) = \frac{\exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle}{1 + \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle} \qquad \text{and} \qquad \widehat{\pi}_0(\boldsymbol{x}) = 1 - \widehat{\pi}_1(\boldsymbol{x}) = \frac{1}{1 + \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x} \rangle}.$$

This provides estimated classifier

$$\widehat{\mathcal{C}}(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; \widehat{\pi}_y(\boldsymbol{x}), \tag{2.28}$$

with a deterministic rule if both $\widehat{\pi}_y(\boldsymbol{x})$ are equally large.

Remarks 2.12.

- Model (2.25)-(2.26) was designed for a binary classification problem with two response classes $J = 2$ (Bernoulli case). Similar results can be derived for multiple response classes $J > 2$ (categorical case).
- The logistic approach (2.27) was used to obtain a probability $p(\boldsymbol{x}) \in (0, 1)$. This is probably the most commonly used approach, but there exist many other (functional) modeling approaches which are in a similar spirit to (2.27).
- In machine learning, the logistic regression assumption (2.27) is also referred to the sigmoid activation function $\phi(x) = (1 + e^{-x})^{-1}$ for $x \in \mathbb{R}$. We come back to this in the neural network chapter, see Table 5.1.
- In categorical problems with more than two classes one often replaces the logistic regression function by the so-called softmax function.
- Note that $\pi_y(\boldsymbol{x})$ is by far more sophisticated than $\mathcal{C}(\boldsymbol{x})$. For instance, if Y is an indicator whether a car driver has an accident or not, i.e.,

$$Y = \mathbb{1}_{\{N \geq 1\}},$$

then we have in the Poisson case

$$\pi_1(\boldsymbol{x}) = p(\boldsymbol{x}) = \mathbb{P}[Y = 1] = \mathbb{P}[N \geq 1] = 1 - \exp\{-\lambda(\boldsymbol{x})v\} = \lambda(\boldsymbol{x})v + o(\lambda(\boldsymbol{x})v), \tag{2.29}$$

as $\lambda(\boldsymbol{x})v \to 0$. Thus, $\pi_1(\boldsymbol{x}) \approx \lambda(\boldsymbol{x})v$ for typical car insurance frequencies, say, of magnitude 5% and one year at risk $v = 1$. This implies that in the low frequency situation we obtain classifier $\mathcal{C}(\boldsymbol{x}) = 0$ for all policies.

Moreover, for small expected frequencies $\lambda(\boldsymbol{x})$ we could also use the logistic regression modeling approach and (2.29) to infer the regression model, we refer to Sections 3.3 and 3.4 in Ferrario et al. [43].

In analogy to the Poisson case we can consider the deviance loss in the binomial case given by

$$\begin{aligned} D^*(\boldsymbol{Y}, \widehat{p}) &= 2\left(\ell_{\boldsymbol{Y}}(\boldsymbol{Y}) - \ell_{\boldsymbol{Y}}(\widehat{\boldsymbol{\beta}})\right) \\ &= -2\sum_{i=1}^{n} Y_i \, \langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle - \log(1 + \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle), \end{aligned} \tag{2.30}$$

because the saturated model provides log-likelihood equal to zero. Similar to (2.15) this allows us for a likelihood ratio test for parameter reduction. It also suggests the Pearson's residuals and the deviance residuals, respectively,

$$\begin{aligned} \widehat{\delta}_i^P &= \frac{Y_i - \widehat{p}(\boldsymbol{x}_i)}{\sqrt{\widehat{p}(\boldsymbol{x}_i)(1 - \widehat{p}(\boldsymbol{x}_i))}}, \\ \widehat{\delta}_i^D &= \operatorname{sgn}(Y_i - 0.5)\sqrt{-2\left(Y_i \, \langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle - \log(1 + \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle)\right)}. \end{aligned}$$

Finally, for out-of-sample back-testing and cross-validation one often considers the probability of misclassification as generalization loss

$$\mathbb{P}\left[Y \neq \widehat{\mathcal{C}}(\boldsymbol{x})\right],$$

for a classifier $\widehat{\mathcal{C}}(\cdot)$ estimated from randomly chosen i.i.d. data $(Y_i, \boldsymbol{x}_i) \sim \mathbb{P}$, $i = 1, \ldots, n$, where the distribution $\mathbb{P}$ has the meaning that we choose at random a policy with feature $\boldsymbol{x}_i$ and corresponding classifier Y_i, and where $(Y, \boldsymbol{x})$ is independent of $\widehat{\mathcal{C}}(\cdot)$ having the same distribution as the cases $(Y_i, \boldsymbol{x}_i)$.
Based on a learning sample $\mathcal{D}^{\mathcal{B}}$ we estimate the classifier by $\widehat{\mathcal{C}}^{\mathcal{B}}(\cdot)$ and using the test sample $\mathcal{D}^{\mathcal{B}^c}$ we can estimate the probability of misclassification defined by

$$\mathcal{L}^{\text{oos}}_{\mathbb{1}_{\{\neq\}}} = \widehat{\mathbb{P}}\left[Y \neq \widehat{\mathcal{C}}(\boldsymbol{x})\right] = \frac{1}{|\mathcal{B}^c|} \sum_{i \in \mathcal{B}^c} \mathbb{1}_{\{Y_i \neq \widehat{\mathcal{C}}^{\mathcal{B}}(\boldsymbol{x}_i)\}}, \tag{2.31}$$

we also refer to Section 1.3.2. Thus, we use the loss function for misclassification

$$L(Y, \mathcal{C}(\boldsymbol{x})) = \mathbb{1}_{\{Y \neq \mathcal{C}(\boldsymbol{x})\}}. \tag{2.32}$$

We can then apply the same techniques as in Section 1.3.2, i.e. the leave-one-out cross-validation or the K-fold cross-validation to estimate this generalization loss.

In relation to Remarks 2.12 one should note that for rare events the misclassification rate should be replaced by other loss functions. In the binary situation with $\pi_1(\boldsymbol{x}) \ll 1/2$ for all $\boldsymbol{x} \in \mathcal{X}$, the trivial predictor $Y \equiv 0$, a.s., obtains an excellent misclassification rate (because of missing sensitivity of this loss function towards rare events); in machine learning this problem is often called class imbalance problem. The binomial deviance loss (2.30) is also referred to the cross-entropy loss which may be used instead of misclassification.

Appendix to Chapter 2

2.6 Maximum likelihood estimation

Under fairly general conditions, the MLE satisfies asymptotic normality properties converging to a multivariate standard Gaussian distribution if properly normalized, see Theorem 6.2.3 in Lehmann [87]. In particular, the MLE is asymptotically unbiased with asymptotic variance described by the inverse of Fisher's information matrix.
The log-likelihood function is in our Poisson GLM given by, see (2.4),

$$\boldsymbol{\beta} \;\mapsto\; \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = \sum_{i=1}^{n} -\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle v_i + N_i\left(\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle + \log v_i\right) - \log(N_i!),$$

and the MLE is found by the roots of

$$\frac{\partial}{\partial \boldsymbol{\beta}}\; \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = 0.$$

Assuming full rank $q+1 \leq n$ of the design matrix $\mathfrak{X}$ we receive a unique solution to this root search problem because under this assumption the log-likelihood function is concave. Fisher's information matrix is in this Poisson GLM given by the negative Hessian, see (2.9),

$$\mathcal{I}(\boldsymbol{\beta}) = \left(-\mathbb{E}\left[\frac{\partial^2}{\partial\beta_l \partial\beta_r}\ell_{\boldsymbol{N}}(\boldsymbol{\beta})\right]\right)_{0\leq l,r\leq q} = \left(\sum_{i=1}^{n} v_i \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle x_{i,l}x_{i,r}\right)_{0\leq l,r\leq q} = -\mathbf{H}_{\boldsymbol{\beta}}\ell_{\boldsymbol{N}}(\boldsymbol{\beta}).$$

Fisher's information matrix can be estimated by

$$\begin{aligned}\mathcal{I}(\widehat{\boldsymbol{\beta}}) \;&=\; \left(\sum_{i=1}^{n} v_i \exp\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i\rangle x_{i,l}x_{i,r}\right)_{0\leq l,r\leq q} \\ &=\; \left(\sum_{i=1}^{n} v_i \exp\left\{(\mathfrak{X}\widehat{\boldsymbol{\beta}})_i\right\} x_{i,l}x_{i,r}\right)_{0\leq l,r\leq q} \;=\; \mathfrak{X}'\, V_{\widehat{\boldsymbol{\beta}}}\, \mathfrak{X},\end{aligned}$$

with estimated diagonal working weight matrix

$$V_{\widehat{\boldsymbol{\beta}}} \;=\; \operatorname{diag}\left(\exp\left\{(\mathfrak{X}\widehat{\boldsymbol{\beta}})_1\right\} v_1, \ldots, \exp\left\{(\mathfrak{X}\widehat{\boldsymbol{\beta}})_n\right\} v_n\right) = \operatorname{diag}\left(V \exp\{\mathfrak{X}\widehat{\boldsymbol{\beta}}\}\right).$$

The inverse of the estimated Fisher's information matrix $\mathcal{I}(\widehat{\boldsymbol{\beta}})^{-1}$ can now be used to estimate the covariance matrix of the estimate $\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}$. In particular, we may use for $0 \leq l \leq q$

$$\widehat{\sigma}_l = \sqrt{\left(\mathcal{I}(\widehat{\boldsymbol{\beta}})^{-1}\right)_{l,l}}\,, \tag{2.33}$$

as the estimated standard error in the MLE $\widehat{\beta}_l$. The diagonal matrix $V_{\widehat{\boldsymbol{\beta}}}$ can be obtained from the R output by using `d.glm$weights` if we set `d.glm <- glm(...)`.

Chapter 3

Generalized Additive Models

In the previous chapter we have assumed that the log-linear model structure (2.2) is appropriate for expected frequency modeling. We have met situations where this log-linear model structure has not really been justified by statistical analysis (feature components `age` or `power`). We have circumvented this problem by building categorical classes and then estimating a regression parameter for each of these categorical classes individually. This approach may lead to over-parametrization and may neglect dependencies between neighboring classes (if there is a natural ordering). In the present section we allow for more flexibility in model assumption (2.2) by considering generalized additive models (GAMs). This chapter is based on Hastie et al. [62], Ohlsson–Johansson [102], Pruscha [106], Wood [134] and Bühlmann–Mächler [20].

3.1 Generalized additive models for Poisson regressions

Assume we are in modeling setup (2.1) and we would like to estimate the regression function $\lambda(\cdot)$. In this chapter we replace the log-linear approach (2.2) by the following model assumption

$$\log \lambda(\boldsymbol{x}) = \beta_0 + \sum_{l=1}^{q} f_l(x_l), \tag{3.1}$$

where $f_l(\cdot)$, $l = 1, \ldots, q$, are sufficiently nice functions and $\beta_0 \in \mathbb{R}$ denotes the intercept.

Remarks 3.1.

- In approach (3.1) we assume that all feature components are real-valued. Categorical components are treated by dummy coding.
- Models of type (3.1) are called generalized additive models (GAMs), here applied to the Poisson case with log-link function (which is the canonical link for the Poisson model). GAMs go back to Hastie–Tibshirani [60, 61], we also refer to Wood [134].
- Approach (3.1) does not allow for non-multiplicative interactions between the feature components. More general versions of GAMs are available in the literature. These may capture other interactions between feature components by, for instance,

considering functions $f_{l_1,l_2}(x_{l_1}, x_{l_2})$ for pair-wise interactions. For interactions between categorical and continuous feature components we also refer to Section 5.6 in [102]. Here, for simplicity, we restrict to models of type (3.1).

- At the moment, the functions $f_l(\cdot)$, $l = 1, \ldots, q$, are not further specified. A necessary requirement to make them uniquely identifiable in the calibration procedure is a normalization condition. For given data $\mathcal{D}$ we require that the functions f_l satisfy for all $l = 1, \ldots, q$
$$\frac{1}{n} \sum_{i=1}^{n} f_l(x_{i,l}) = 0. \tag{3.2}$$

- The log-link in (3.1) leads to a multiplicative tariff structure
$$\lambda(\boldsymbol{x}) = \exp\{\beta_0\} \prod_{l=1}^{q} \exp\{f_l(x_l)\}. \tag{3.3}$$
Normalization (3.2) then implies that $\exp\{\beta_0\}$ describes the base premium and we obtain multiplicative correction factors $\exp\{f_l(x_l)\}$ relative to this base premium for features $\boldsymbol{x} \in \mathcal{X}$, we also refer to (2.10).

In the sequel of this section we consider *natural cubic splines* for the modeling of f_l.

3.1.1 Natural cubic splines

A popular approach is to use natural cubic splines for the functions $f_l(\cdot)$ in (3.1). Choose a set of m knots $u_1 < \ldots < u_m$ on the real line $\mathbb{R}$ and define the function $f : [u_1, u_m] \to \mathbb{R}$ as follows:
$$f(x) = h_t(x), \qquad \text{for } x \in [u_t, u_{t+1}), \tag{3.4}$$
where for $t = 1, \ldots, m-1$ we choose cubic functions $h_t(x) = \alpha_t + \vartheta_t x + \gamma_t x^2 + \delta_t x^3$ on $\mathbb{R}$. For the last index $t = m-1$ we extend (3.4) to the closed interval $[u_{m-1}, u_m]$.
We say that the function f defined in (3.4) is a *cubic spline* if it satisfies the following differentiability (smoothing) conditions in the (internal) knots $u_2 < \ldots < u_{m-1}$
$$h_{t-1}(u_t) = h_t(u_t), \qquad h'_{t-1}(u_t) = h'_t(u_t) \qquad \text{and} \qquad h''_{t-1}(u_t) = h''_t(u_t), \tag{3.5}$$
for all $t = 2, \ldots, m-1$. Function f has $4(m-1)$ parameters and the constraints (3.5) reduce these by $3(m-2)$. Therefore, a cubic spline has $m+2$ degrees of freedom.
Observe that (3.5) implies that a cubic spline is twice continuously differentiable on the interval (u_1, u_m). If we extend this cubic spline f twice continuously differentiable to an interval $[a, b] \supset [u_1, u_m]$ with linear extensions on $[a, u_1]$ and $[u_m, b]$, we call this spline *natural cubic spline*. This provides two additional boundary constraints $f''(x) = 0$ on $[a, u_1] \cup [u_m, b]$, thus, $h''_1(u_1) = h''_{m-1}(u_m) = 0$, and reduces the degrees of freedom by 2. Therefore, a natural cubic spline has m degrees of freedom.
Note that a natural cubic spline can be represented by functions $x \mapsto (x - u_t)_+^3$, $t = 1, \ldots, m$. Namely,
$$f(x) = \alpha_0 + \vartheta_0 x + \sum_{t=1}^{m} c_t (x - u_t)_+^3, \qquad \text{with } \sum_{t=1}^{m} c_t = 0 \text{ and } \sum_{t=1}^{m} c_t u_t = 0, \tag{3.6}$$

gives a natural cubic spline. The two side constraints ensure that we have a smooth linear extension above u_m. This again provides m degrees of freedom and these natural cubic splines (with m given knots) build an m-dimensional linear space. The knots $u_1 < \ldots < u_m$ play a particularly special role: assume that in these knots we are given values $f_1^*, \ldots, f_m^*$. Then, there exists a unique natural cubic spline f on $[a, b]$ with $f(u_t) = f_t^*$ for all $t = 1, \ldots, m$, see Theorem 5.1 in [102].

We would like to solve the following optimization problem:

Find the intercept β_0 and the natural cubic splines $f_1, \ldots, f_q$ satisfying normalization conditions (3.2) such that the following expression is *minimized*:

$$D^*(\boldsymbol{N}, \lambda) + \sum_{l=1}^{q} \eta_l \int_{a_l}^{b_l} \left(f_l''(x_l)\right)^2 dx_l, \tag{3.7}$$

with observations $\boldsymbol{N} = (N_1, \ldots, N_n)'$, intercept and natural cubic splines given by $\boldsymbol{f} = (\beta_0; f_1, \ldots, f_q)$ and determining the GAM regression function $\lambda(\cdot)$ by (3.1), tuning parameters $\eta_l \geq 0$, $l = 1, \ldots, q$, and Poisson deviance loss given by

$$D^*(\boldsymbol{N}, \lambda) \;=\; \sum_{i=1}^{n} 2\, N_i \left[\frac{\lambda(\boldsymbol{x}_i) v_i}{N_i} - 1 - \log\left(\frac{\lambda(\boldsymbol{x}_i) v_i}{N_i} \right) \right],$$

where the right-hand side is set equal to $2\lambda(\boldsymbol{x}_i)v_i$ for $N_i = 0$. The supports $[a_l, b_l]$ of the natural cubic splines f_l should contain all observed feature components $x_{i,l}$ of $\mathcal{D}$.

Remarks 3.2.

- We refer to page 35 for the interplay between maximizing the log-likelihood function and minimizing the deviance loss. In short, we are minimizing the in-sample loss in (3.7), modeled by the deviance loss $D^*(\boldsymbol{N}, \lambda)$, which is equivalent to maximizing the corresponding log-likelihood function, subject to regularization conditions that we discuss next.

- The regularization conditions for the natural cubic splines f_l

$$\int_{a_l}^{b_l} \left(f_l''(x_l)\right)^2 dx_l = \int_{u_{1,l}}^{u_{m_l,l}} \left(f_l''(x_l)\right)^2 dx_l \tag{3.8}$$

  guarantee in the optimization of (3.7) that the resulting (optimal) functions f_l do not look too wildly over their domains $[a_l, b_l] \supset [u_{1,l}, u_{m_l,l}]$, where $u_{1,l}, \ldots, u_{m_l,l}$ are the m_l knots of f_l. In particular, we want to prevent from over-fitting to the data $\mathcal{D}$. The tuning parameters $\eta_l \geq 0$ balance the influence of the regularization conditions (3.8). Appropriate tuning parameters are (often) determined by cross-validation.

- Usually, one starts from a more general optimization problem, namely, minimize

$$\begin{aligned} D^*(\boldsymbol{N},\lambda) \quad &+ \quad \sum_{l=1}^{q} \eta_l \int_{a_l}^{b_l} \left(f_l''(x_l)\right)^2 dx_l \\ &= \sum_{i=1}^{n} 2\left[e^{\beta_0+\sum_{l=1}^{q} f_l(x_{i,l})} v_i - N_i - N_i\left(\beta_0 + \sum_{l=1}^{q} f_l(x_{i,l})\right) + N_i \log(N_i/v_i)\right] \\ &\qquad + \sum_{l=1}^{q} \eta_l \int_{a_l}^{b_l} \left(f_l''(x_l)\right)^2 dx_l \end{aligned} \tag{3.9}$$

  over $\beta_0 \in \mathbb{R}$ and over all (normalized) functions $f_1, \ldots, f_q$ that are twice continuously differentiable on their domains.

  $\triangleright$ Assume that the feature component x_l has exactly $m_l \leq n$ different values $x^*_{1,l} < \ldots < x^*_{m_l,l}$ among the observations $\mathcal{D}$. This implies that the deviance loss $D^*(\boldsymbol{N},\lambda)$ only depends on these values $f^*_{i,l} = f_l(x^*_{i,l})$ for $i = 1, \ldots, m_l$ and $l = 1, \ldots, q$, see middle line of (3.9).

  $\triangleright$ Theorem 5.1 (part 1) in [102] says that for given values $(x^*_{i,l}, f^*_{i,l})_{i=1,\ldots,m_l}$ there exists a *unique natural cubic spline* f^*_l on $[a_l, b_l]$ with $f^*_l(x^*_{i,l}) = f^*_{i,l}$ for all $i = 1, \ldots, m_l$. Thus, a choice $(x^*_{i,l}, f^*_{i,l})_{i=1,\ldots,m_l}$ completely determines the natural cubic spline f^*_l.

  $\triangleright$ Theorem 5.1 (part 2) in [102] says that any twice continuously differentiable function f_l with $f_l(x^*_{i,l}) = f^*_{i,l}$ for all $i = 1, \ldots, m_l$, has a regularization term (3.8) that is at least as big as the one of the corresponding natural cubic spline f^*_l.

  For these reasons we can restrict the (regularized) optimization to natural cubic splines. This basically means that we need to find the optimal values $(f^*_{i,l})_{i=1,\ldots,m_l}$ in the feature components/knots $(x^*_{i,l})_{i=1,\ldots,m_l}$ subject to regularization conditions (3.8) for given tuning parameters $\eta_l \geq 0$ and normalization (3.2) for $l = 1, \ldots, q$.

- Note that $m_l < n$ is quite common, for instance, our $n = 500'000$ car drivers only have $m_l = 90 - 18 + 1 = 73$ different drivers' ages, see Appendix A. In this case, the deviance loss $D^*(\boldsymbol{N},\lambda)$ is for feature component x_l completely determined by the values $f^*_{i,l} = f_l(x^*_{i,l})$, $i = 1, \ldots, m_l = 73$. For computational reasons one may even merge more of the feature component values (by rounding/bucketing) to get less values $f^*_{i,l}$ and knots $u_{i,l} = x^*_{i,l}$ in the optimization.

Since natural cubic splines with given knots $x^*_{1,l}, \ldots, x^*_{m_l,l}$ build an m_l-dimensional linear space, we may choose m_l linearly independent basis functions $s_{1,l}, \ldots, s_{m_l,l}$ (of the form (3.6)) and represent the natural cubic splines f_l on $[a_l, b_l]$ by

$$f_l(x_l) = \sum_{k=1}^{m_l} \beta_{k,l} s_{k,l}(x_l), \tag{3.10}$$

for unique constants $\beta_{1,l}, \ldots, \beta_{m_l,l}$. Observe that (3.10) brings us fairly close to the GLM framework of Chapter 2. Assume that all natural cubic splines f_l are represented in the form (3.10), having parameters $\beta_{1,l}, \ldots, \beta_{m_l,l}$ and corresponding basis functions

$s_{1,l}, \ldots, s_{m_l,l}$, then we can rewrite (3.1) as

$$\log \lambda(\boldsymbol{x}) = \beta_0 + \sum_{l=1}^{q} \sum_{k=1}^{m_l} \beta_{k,l} s_{k,l}(x_l) \overset{\text{def.}}{=} \langle \boldsymbol{\beta}, \boldsymbol{s}(\boldsymbol{x}) \rangle, \tag{3.11}$$

where the last identity defines the scalar product between $\boldsymbol{\beta} = (\beta_0, \beta_{1,1}, \ldots, \beta_{m_q,q})'$ and $\boldsymbol{s}(\boldsymbol{x}) = (1, s_{1,1}(x_1), \ldots, s_{m_q,q}(x_q))' \in \mathbb{R}^{r+1}$. Moreover, note that for these natural cubic splines we have

$$\begin{aligned} \int_{a_l}^{b_l} \left(f_l''(x_l)\right)^2 dx_l &= \int_{u_{1,l}}^{u_{m_l,l}} \left(\sum_{k=1}^{m_l} \beta_{k,l} s_{k,l}''(x_l) \right)^2 dx_l \\ &= \sum_{k,j=1}^{m_l} \beta_{k,l}\beta_{j,l} \int_{u_{1,l}}^{u_{m_l,l}} s_{k,l}''(x_l) s_{j,l}''(x_l) dx_l \overset{\text{def.}}{=} \sum_{k,j=1}^{m_l} \beta_{k,l}\beta_{j,l}\, \omega_{k,j}^{(l)}. \end{aligned}$$

Therefore, optimization problem (3.7) is transformed to (we drop the irrelevant terms):

Minimize over $\boldsymbol{\beta} \in \mathbb{R}^{r+1}$ the objective function

$$\sum_{i=1}^{n} 2 \left[\exp \langle \boldsymbol{\beta}, \boldsymbol{s}(\boldsymbol{x}_i) \rangle v_i - N_i \langle \boldsymbol{\beta}, \boldsymbol{s}(\boldsymbol{x}_i) \rangle \right] + \boldsymbol{\beta}'\, \Omega(\boldsymbol{\eta})\, \boldsymbol{\beta}, \tag{3.12}$$

with block-diagonal matrix $\Omega(\boldsymbol{\eta}) = \operatorname{diag}(0, \Omega_1, \ldots, \Omega_d) \in \mathbb{R}^{(r+1)\times(r+1)}$ having blocks

$$\Omega_l = \Omega_l(\eta_l) = \eta_l \left(\omega_{k,j}^{(l)}\right)_{k,j=1,\ldots,m_l} \in \mathbb{R}^{m_l \times m_l},$$

for $l = 1, \ldots, q$, tuning parameters $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_q)'$, and under side constraints (3.2).

The optimal parameter $\widehat{\boldsymbol{\beta}}$ of this optimization problem is found numerically.

Remarks 3.3.

- The normalization conditions (3.2) provide identifiability of the parameters:

$$0 = \sum_{i=1}^{n} f_l(x_{i,l}) = \sum_{i=1}^{n} \sum_{k=1}^{m_l} \beta_{k,l} s_{k,l}(x_{i,l}) = \sum_{k=1}^{m_l} \left(\beta_{k,l} \sum_{i=1}^{n} s_{k,l}(x_{i,l}) \right).$$

  Often these conditions are not explored at this stage, but functions are only adjusted at the very end of the procedure. In particular, we first calculate a relative estimator and the absolute level is only determined at the final stage of the calibration. This relative estimator can be determined by the *back-fitting algorithm for additive models*, for details see Algorithm 9.1 in Hastie et al. [62] and Section 5.4.2 in Ohlsson–Johansson [102].

- A second important remark is that the calculation can be accelerated substantially if one "rounds" the feature components, i.e., for instance, for the feature component `dens` we may choose the units in hundreds, which substantially reduces m_l and, hence, the computational complexity, because we obtain less knots and hence a lower dimensional $\boldsymbol{\beta}$. Often one does not lose (much) accuracy by this rounding and, in particular, one is less in the situation of a potential over-parametrization and over-fitting, we also refer to Section 2.4.3 on data compression.

- In the example below, we use the command `gam` from the R package `mgcv`.

3.1.2 Example in motor insurance pricing, revisited

We present two applications of the GAM approach. We revisit the car insurance example of Section 2.4 which is based on the data generated in Appendix A. The first application only considers the two feature components `age` and `ac`, this is similar to Example 2.10. The second application considers all feature components, similarly to Section 2.4.5.

Example 3.4 (example GAM1)**.** We consider the same set-up as in Example 2.10 (GLM1) but we model the feature components `age` and `ac` by natural cubic splines. Note that we have $m_1 = 73$ different ages of drivers and $m_2 = 36$ different ages of cars. For computational reasons, it is important in GAM calibrations that data is compressed accordingly, as described in Section 2.4.3. This gives us at most $m_1 m_2 = 73 \cdot 36 = 2'628$ different risk cells. In our data $\mathcal{D}$, only $2'223$ of these risk cells are non-empty, i.e. have a volume $v_k^+ > 0$, for the latter see also (2.18). Thus, the data is reduced from $n = 500'000$ observations to a sufficient statistics of size $d = 2'223$.[1]

Listing 3.1: Results of example GAM1

```
Family: poisson
Link function: log

Formula:
claims ~ s(age, bs = "cr", k = k1) + s(ac, bs = "cr", k = k2)

Parametric coefficients:
             Estimate Std. Error z value Pr(>!z!)
(Intercept) -2.39356    0.02566  -93.29   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Approximate significance of smooth terms:
         edf Ref.df Chi.sq p-value
s(age) 12.48  15.64   1162  <2e-16 ***
s(ac)  17.43  20.89   3689  <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

R-sq.(adj) =  0.963   Deviance explained =   65%
UBRE = -0.012852  Scale est. = 1         n = 2223
```

The natural cubic splines approach is implemented in the command `gam` of the R package `mgcv`, and the corresponding results are provided in Listing 3.1. The feature components `s(age,bs="cr",k=k1)` and `s(di,bs="cr",k=k2)` are being modeled as continuous variables (indicated by `s(·)`) and we fit cubic splines (indicated by `bs="cr"`). The parameters `k1` $= m_1 = 73$ and `k2` $= m_2 = 36$ indicate how many knots we would like to have for each of the two feature components `age` and `ac`. Note that we choose the maximal number of possible knots (different labels) here, see also Remarks 3.2. This number is usually too large and one should choose a lower number of knots for computational reasons. We did not specify the tuning parameters η_l in Listing 3.1. If we drop these tuning parameters in the R command `gam`, then a generalized cross-validation (GCV) criterion or an unbiased risk estimator (UBRE) criterion is applied to determine good tuning parameters

[1] This data compression reduces the run time of the GAM calibration from 1'018 seconds to 1 second!

(internally). The GCV criterion considers a scaled in-sample loss given by

$$\mathrm{GCV}(\boldsymbol{\eta}) = (1 - M(\boldsymbol{\eta})/n)^{-2}\, \mathcal{L}_D^{\mathrm{is}} = \left(\frac{n}{n - M(\boldsymbol{\eta})}\right)^2 \mathcal{L}_D^{\mathrm{is}}, \tag{3.13}$$

where $M(\boldsymbol{\eta})$ is the effective degrees of freedom of the model. This effective degrees of freedom is obtained from the corresponding influence matrix, for more details we refer to Wood [134], Section 3.2.3, and Hastie et al. [62], Section 7.10.1. The GCV criterion has the advantage over K-fold cross-validation that it is computationally much more fast, which is important in the optimization algorithm. We can extract the resulting optimal tuning parameters with the command `gam$sp` which provides $(\eta_1, \eta_2) = (273'401, 2'591)$ in our example. The column `edf` in Listing 3.1 shows the effective degrees of freedom which corresponds to the number of knots needed (for this optimal tuning parameters), for details we refer to Section 5.4.1 in Hastie et al. [62]. If `edf` is close to 1 then we basically fit a straight line which means that we can use the GLM (log-linear form) for this feature component. The last two columns on lines 15-16 of Listing 3.1 give an approximate χ^2-test for assessing the significance of the corresponding smooth term. The corresponding p-values are roughly zero which says that we should keep both feature components.

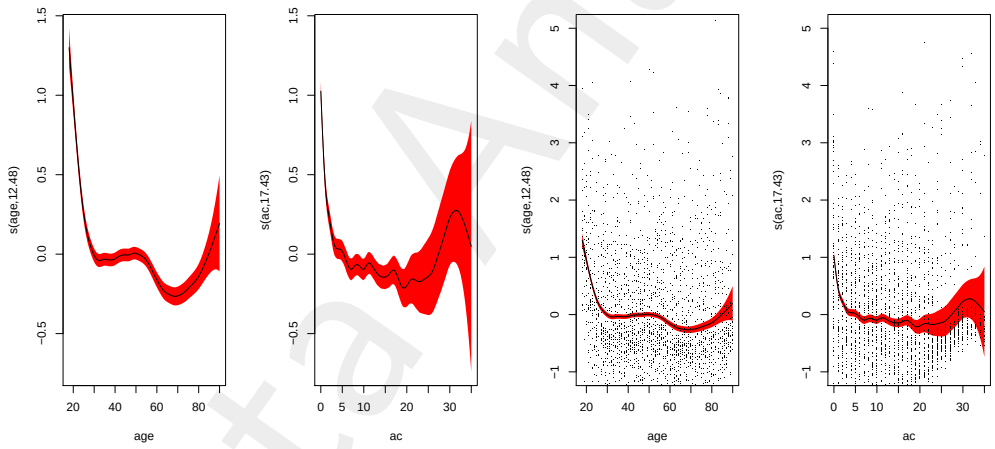


Figure 3.1: GAM1 results with $(\eta_1, \eta_2) = (273'401, 2'591)$, and `k1` $= 73$ and `k2` $= 36$; (lhs) fitted splines `s(age,bs="cr",k=k1)` and `s(di,bs="cr",k=k2)` excluding observations, and (rhs) including observations; note that (η_1, η_2) is determined by GCV here.

In Figure 3.1 we provide the resulting marginal plots for `age` and `ac`. These are excluding (lhs) and including (rhs) the observations (and therefore have different scales on the y-axis). Moreover, we provide confidence bounds in red color. From these plots it seems that `age` is fitted well, but that `ac` is over-fitting for higher ages of the car, thus, one should either decrease the number of knots `k2` or increase the tuning parameter η_2.
In Figure 3.2 we provide the resulting two dimensional surface (from two different angles). We observe a rather step slope for small `age` and small `ac` which reflects the observed frequency behavior.

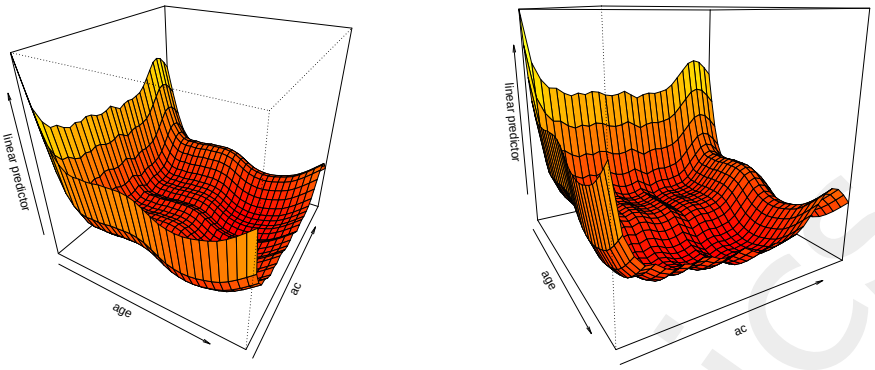


Figure 3.2: GAM1 results with $(\eta_1, \eta_2) = (273'401, 2'591)$, and `k1` $= 73$ and `k2` $= 36$ from two different angles.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.1) GLM1	3.1s	11	28.3543	28.3544	0.6052	28.3510	10.2691%
(Ch3.1) GAM1	1.1s	108	28.3248	28.3245	0.5722	28.3134	10.2691%

Table 3.1: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 2.4.

In Table 3.1 we provide the corresponding prediction results on line (Ch3.1). The number of parameters '# param.' is determined by the number of chosen knots minus one, i.e. `k1` $+$ `k2` $- 1 = m_1 + m_2 - 1 = 73 + 36 - 1 = 108$, where the minus one corresponds to the intercept parameter minus two normalization conditions (3.2). We observe a smaller in-sample loss $\mathcal{L}_D^{\text{is}}$ of GAM1 compared to GLM1, i.e. the GAM can better adapt to the data $\mathcal{D}$ than the GLM (considering the same feature components `age` and `ac`). This shows that the choices of the categorical classes for `age` and `ac` can be improved in the GLM approach. This better performance of the GAM carries over to the cross-validation losses $\mathcal{L}_D^{\text{CV}}$ and the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$. We also note that the difference between the in-sample loss and the cross-validation loss increases, which shows that the GAM has a higher potential for over-fitting than the GLM, here.
In Figure 3.3 we present the results for different choices of the tuning parameters η_l. For small tuning parameters ($\eta_l = 10$ in our situation) we obtain wiggly pictures which follow closely the observations. For large tuning parameters ($\eta_l = 10'000'000$ in our situation) we obtain rather smooth graphs with small degrees of freedom, see Figure 3.3 (rhs). Thus, we may get either over-fitting (lhs) and under-fitting (rhs) to the data.

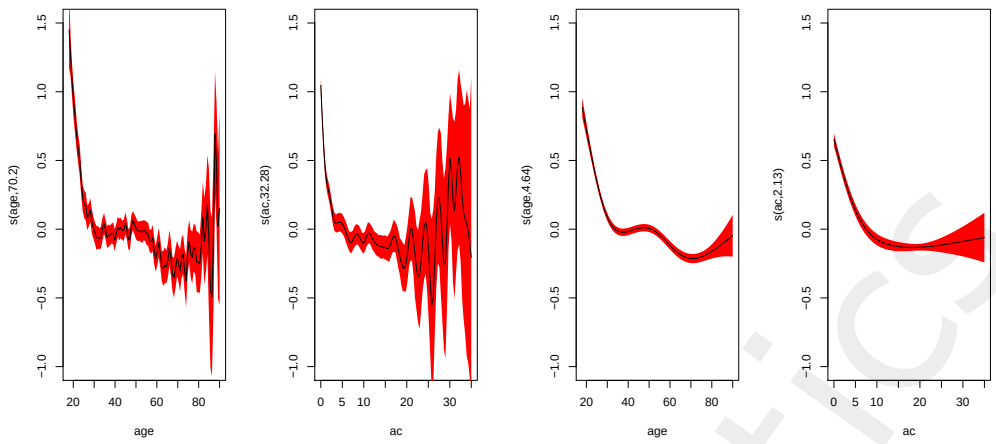


Figure 3.3: GAM1 results with (lhs) $(\eta_1, \eta_2) = (10, 10)$ and (rhs) $(\eta_1, \eta_2) = (10'000'000, 10'000'000)$ for `k1` $= 73$ and `k2` $= 36$.

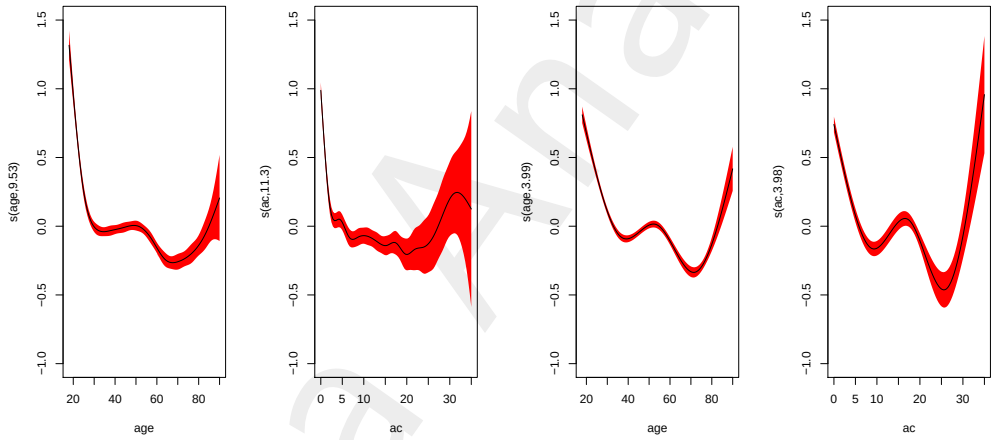


Figure 3.4: GAM1 results with number of knots (lhs) `k1` $= 12$ and `k2` $= 17$ and (rhs) `k1` $= 5$ and `k2` $= 5$; note that (η_1, η_2) is determined by GCV here.

In Figure 3.4 we change the number of knots (`k1`, `k2`) and we let the optimal tuning parameters be determined using the GCV criterion. We note that for less knots the picture starts to smooth and the confidence bounds get more narrow because we have less parameters to be estimated. For one knot we receive a log-linear GLM component. Finally, we evaluate the quality of the GCV criterion. From Listing 3.1 we see that the effective degrees of freedom are $M(\boldsymbol{\eta}) = 1 + 12.48 + 17.43 = 30.91$. Thus, we obtain

$$\text{GCV}(\boldsymbol{\eta}) = \left(\frac{n}{n - M(\boldsymbol{\eta})}\right)^2 \mathcal{L}_D^{\text{is}} = \left(\frac{500'000}{500'000 - 30.91}\right)^2 28.3134 \cdot 10^{-2} = 28.3166 \cdot 10^{-2}.$$

This is smaller than the K-fold cross-validation errors $\mathcal{L}_D^{\text{CV}}$ on line (Ch3.1) of Table 3.1.

This indicates that GCV is likely to under-estimate over-fitting here. This closes our first GAM example. ■

Example 3.5 (example GAM2/3)**.** In our second example we consider all available feature components. We keep `gas`, `brand` and `ct` as categorical. All other feature components are modeled with natural cubic splines, using the GCV criterion to determine the optimal tuning parameters η_l. To keep computational time under control we use data compression, and before that we modify $\log(\texttt{dens})/2$ to be rounded to one digit (which gives 48 different values). The data is then compressed correspondingly leading to $d = 235'163$ compressed observations.[2] This compression gives us $73 + 36 + 12 + 2 + 11 + 6 + 48 + 26 + 1 - 8 = 207$ parameters to be estimated. This should be compared to Conclusion 2.8.

Listing 3.2: Results of example GAM2

```
1  Family: poisson
2  Link function: log
3
4  Formula:
5  claims ~ s(age, bs="cr", k=73) + s(ac, bs="cr", k=36) + s(power, bs="cr", k=12)
6           + gas + brand + s(area, bs="cr", k=6) + s(densGAM, bs="cr", k=48) + ct
7
8  Parametric coefficients:
9               Estimate Std. Error  z value Pr(>!z!)
10 (Intercept) -2.254150   0.020420 -110.391  < 2e-16 ***
11 gasRegular   0.073706   0.013506    5.457 4.83e-08 ***
12 brandB10     0.044510   0.045052    0.988 0.323174
13 .
14 brandB5      0.107430   0.029505    3.641 0.000271 ***
15 brandB6      0.020794   0.033406    0.622 0.533627
16 ctAG        -0.094460   0.027672   -3.414 0.000641 ***
17 .
18 ctZG        -0.079805   0.046513   -1.716 0.086208 .
19 ---
20 Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
21
22 Approximate significance of smooth terms:
23               edf Ref.df   Chi.sq p-value
24 s(age)     13.132 16.453 1161.419  <2e-16 ***
25 s(ac)      17.680 21.147 2699.012  <2e-16 ***
26 s(power)    9.731 10.461  311.483  <2e-16 ***
27 s(area)     1.009  1.016    5.815  0.0164 *
28 s(densGAM)  5.595  7.071   10.251  0.1865
29 ---
30 Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
31
32 R-sq.(adj) =  0.115   Deviance explained = 4.74%
33 UBRE = -0.56625  Scale est. = 1         n = 235163
```

The estimation results are presented in Listing 3.2. The upper part (lines 10-18) shows the intercept estimate $\widehat{\beta}_0$ as well as the estimates for the categorical variables `gas`, `brand` and `ct`. A brief inspection of these numbers shows that we keep all these variables in the model.

[2] The data compression takes 44 seconds.

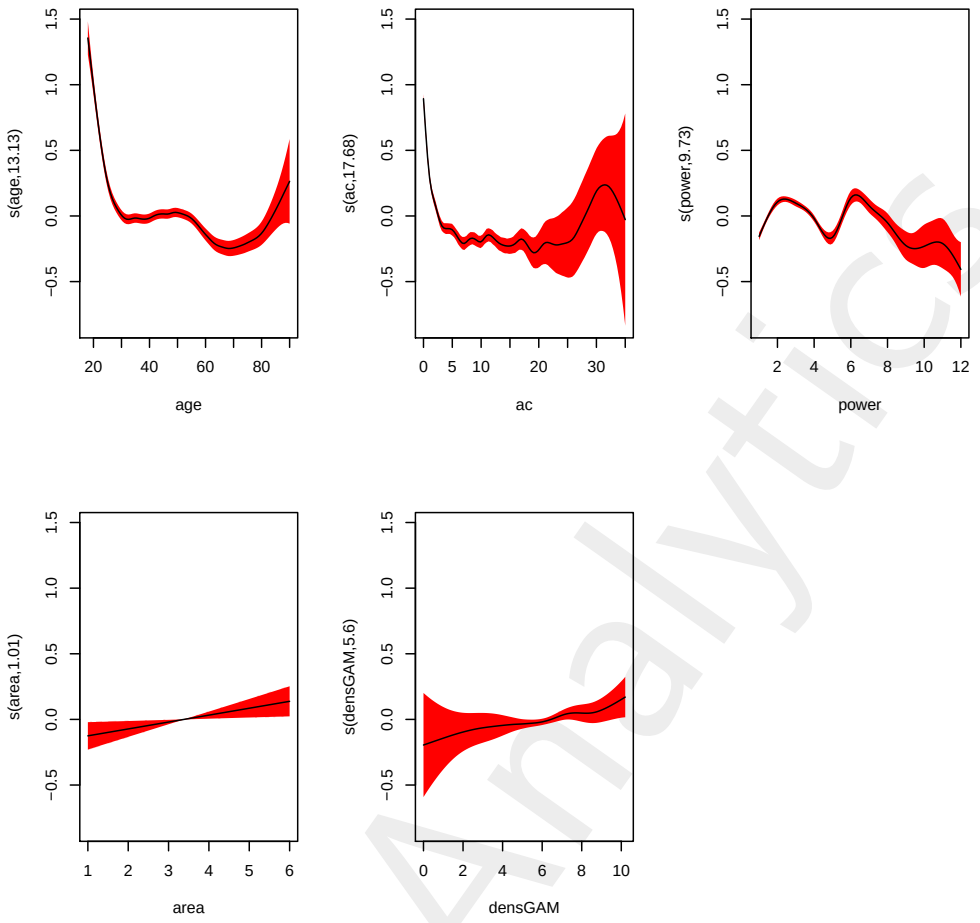


Figure 3.5: GAM2 results: marginal splines approximation of the continuous feature components `age`, `ac`, `power`, `area` and log(`dens`).

The lower part of Listing 3.2 (lines 24-28) shows the results of the 5 continuous feature components `age`, `ac`, `power`, `area` and log(`dens`). The first 3 continuous feature components should be included in this natural cubic spline form. The effective degrees of freedom of the feature component `area` is very close to 1, which suggests that this component should be modeled in log-linear form. Finally, the component log(`dens`) does not seem to be significant which means that we may drop it from the model. In Figure 3.5 we show the resulting marginal spline approximations, which confirm the findings of Listing 3.2, in particular, `area` is log-linear whereas log(`dens`) can be modeled by almost a horizontal line (respecting the red confidence bounds and keeping in mind that `area` and `dens` are strongly positively correlated, see Table A.2).

In Table 3.2 we provide the resulting losses of model GAM2 on line (Ch3.2). Firstly, we see that the run time of the calibration takes roughly 10 minutes, i.e. quite long. This run time could be reduced if we choose less knots in the natural cubic splines. Secondly, we observe that the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ and the in-sample loss $\mathcal{L}_D^{\text{is}}$ become smaller

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.2) GAM2	678s	207	–	–	0.3877	28.0927	10.2690%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%

Table 3.2: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 2.5.

compared to model GLM4 which shows that the categorization in GLM4 is not optimal. We refrain from calculating the cross-validation losses for model GAM2 because this takes too much time. Finally, in Figure 3.6 we illustrate the resulting two-dimensional surfaces of the continuous feature components `ac`, `power`, `age` and log(`dens`).

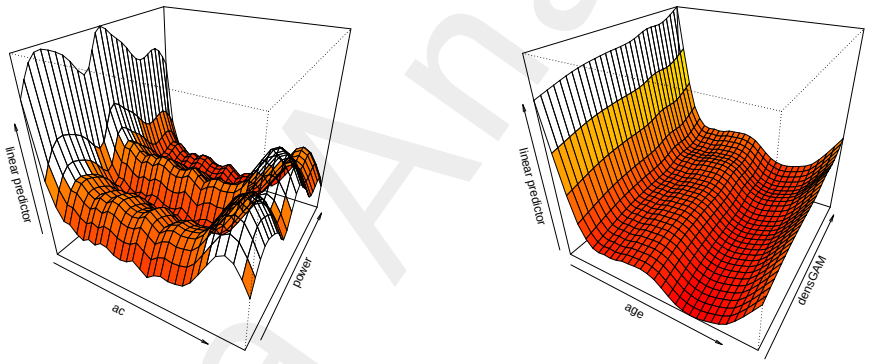


Figure 3.6: GAM2 results of the continuous components `ac`, `power`, `age` and log(`dens`).

In view of Listing 3.2 we may consider a simplified GAM. We model the feature components `age` and `ac` by natural cubic splines having `k1` = 14 and `k2` = 18 knots, respectively, because the corresponding effective degrees of freedom in Listing 3.2 are 13.132 and 17.680 (the tuning parameters η_l are taken equal to zero for these two components). The component `power` is transformed to a categorical variable because the effective degrees of freedom of 9.731 is almost equal to the number of levels (these are 11 parameters under dummy coding), suggesting that each label should have its own regression parameter β_l. The component `area` is considered in log-linear form, the component `dens` is dropped from the model, and `gas`, `brand` and `ct` are considered as categorical variables. We call this model GAM3. The calibration of this GAM takes roughly 50 seconds, which makes it feasible to perform K-fold cross-validation.

Listing 3.3: Results of example GAM3

```
Family: poisson
Link function: log

Formula:
claims ~ s(age, bs = "cr", k = 14) + s(ac, bs = "cr", k = 18)
         + powerCat + gas + brand + area + ct

Parametric coefficients:
             Estimate Std. Error z value Pr(>!z!)
(Intercept) -2.674145   0.032674 -81.844  < 2e-16 ***
powerGAM2    0.264250   0.021278  12.419  < 2e-16 ***
.
powerGAM12  -0.275665   0.109984  -2.506 0.012197 *
gasRegular   0.072807   0.013526   5.383 7.34e-08 ***
brandB10     0.045986   0.045066   1.020 0.307524
.
brandB5      0.107218   0.029511   3.633 0.000280 ***
brandB6      0.021155   0.033407   0.633 0.526560
area         0.084586   0.005360  15.782  < 2e-16 ***
ctAG        -0.102839   0.027335  -3.762 0.000168 ***
.
ctZG        -0.085552   0.046349  -1.846 0.064917 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Approximate significance of smooth terms:
       edf Ref.df Chi.sq p-value
s(age)  13     13   1192  <2e-16 ***
s(ac)   17     17   2633  <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

R-sq.(adj) =  0.0334   Deviance explained = 3.44%
UBRE = -0.71863  Scale est. = 1         n = 500000
```

The results are presented on line (Ch3.3) of Table 3.2 and in Listing 3.3 (we do not choose any regularization here because the chosen numbers of knots are assumed to be optimal). We conclude that this model GAM3 has a clearly better performance than model GLM4. It is slightly worse than GAM2, however, this comes at a much lower run time. For this reason we prefer model GAM3 in all subsequent considerations.

In Figure 3.7 (lhs) we provide the scatter plot of the resulting estimated frequencies (on the log scale) between the two models GLM4 and GAM3. The colors illustrate the years at risk v_i on a policy level. The cloud of frequencies lies on the 45 degrees axis which indicates that model GLM4 and model GAM3 make similar predictions $\widehat{\lambda}(\boldsymbol{x}_i)$ on a policy level. In Figure 3.7 (middle and rhs) we compare these predictions to the true frequencies $\lambda^\star(\boldsymbol{x}_i)$ (which are available for our synthetic data). These two clouds have a much wider diameter which says that the models GLM4 and GAM3 have quite some room for improvements. This is what we are going to explore in the next chapters. This finishes the GAM example. ■

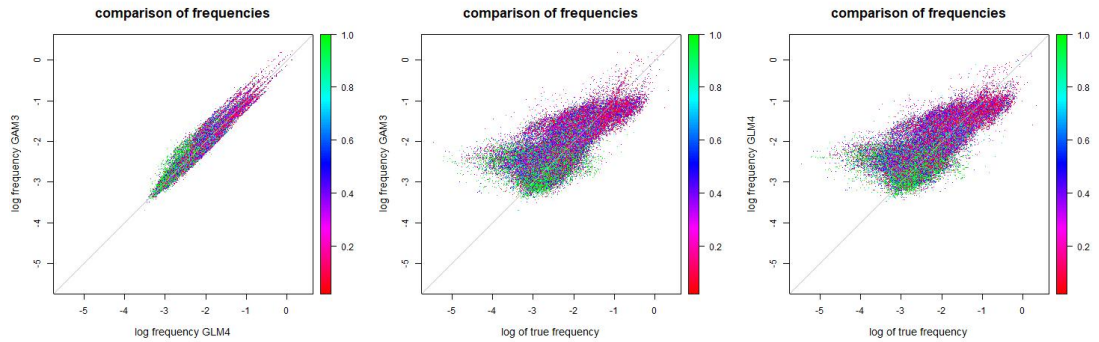


Figure 3.7: Resulting estimated frequencies (on the log scale) of models GLM4 and GAM3, also compared to the true frequencies.

3.1.3 Generalized additive models: summary

We have seen that GAMs provide an improvement over GLMs for non log-linear feature components. In particular, GAMs make it redundant to categorize non log-linear feature components in GLM applications. This has the advantage that the ordinal relationship is kept. The drawback of GAMs is that they are computationally more expensive than GLMs.

We have not explored interactions beyond multiplicative structures, yet. This may be a main reason why the comparisons in Figure 3.7 (middle and rhs) do not look fully convincing. Having 500'000 observation we could directly explore such interactions in GLMs and GAMs, see also Remarks 2.9. We refrain from doing so but we will provide other data driven methods below.

3.2 Multivariate adaptive regression splines

The technique of multivariate adaptive regression splines (MARS) uses piece-wise linear basis functions. These piece-wise linear basis functions have the following two forms on $\mathcal{X} \subset \mathbb{R}^q$:

$$\boldsymbol{x} \mapsto h_{-t,l}(\boldsymbol{x}) = (x_l - t)_+ \qquad \text{or} \qquad \boldsymbol{x} \mapsto h_{+t,l}(\boldsymbol{x}) = (t - x_l)_+, \tag{3.14}$$

for a given constant t and a given feature component x_l. The pair in (3.14) is called *reflected pair* with knot t for feature component x_l. These reflected pairs consist of so-called rectified linear unit (ReLU) or hinge functions. We define the set of all such reflected pairs that are generated by the features in $\mathcal{D}$. These are given by the basis

$$\mathcal{H} = \{h_{-t_l,l}(\cdot), h_{+t_l,l}(\cdot);\ l = 1, \ldots, q \text{ and } t_l = x_{1,l}, \ldots, x_{n,l}\}\,.$$

Note that $\mathcal{H}$ spans a space of continuous piece-wise linear functions (splines). MARS makes the following modeling approach: choose the expected frequency function as

$$\boldsymbol{x} \ \mapsto \ \log \lambda(\boldsymbol{x}) \ = \ \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}), \tag{3.15}$$

for given $M \geq 0$, and $h_m(\cdot)$ are functions from the basis $\mathcal{H}$ or products of such basis functions in $\mathcal{H}$. The latter implies that the non-zero part of $h_m(\cdot)$ can also be non-linear.

Observe that this approach is rather similar to the GAM approach with natural cubic splines. The natural cubic splines are generated by functions $\boldsymbol{x} \mapsto (x_l - t)_+^3$ with $t = x_{i,l}$, see (3.6), which can be obtained by multiplying the basis function $h_{-t,l}(\boldsymbol{x}) \in \mathcal{H}$ three times with itself.

This model is fit by an iterative (stage-wise adaptive) algorithm that selects at each step a function $h_m(\cdot)$ of the present calibration, and splits this function into the two (new) functions

$$\boldsymbol{x} \mapsto h_{m_1}(\boldsymbol{x}) = h_m(\boldsymbol{x}) \, (x_l - x_{i,l})_+ \qquad \text{and} \qquad \boldsymbol{x} \mapsto h_{m_2}(\boldsymbol{x}) = h_m(\boldsymbol{x}) \, (x_{i,l} - x_l)_+ \,,$$

for some $l = 1, \ldots, q$ and $i = 1, \ldots, n$. Thereby, it chooses the additional optimal split (in terms of m, l and $x_{i,l}$) to obtain the new (refined) model

$$\begin{aligned} \log \lambda(\boldsymbol{x}) &= \beta_0 + \sum_{m' \neq m} \beta_{m'} h_{m'}(\boldsymbol{x}) \; + \; \beta_{m_1} h_m(\boldsymbol{x}) h_{-x_{i,l},l}(\boldsymbol{x}) \; + \; \beta_{m_2} h_m(\boldsymbol{x}) h_{+x_{i,l},l}(\boldsymbol{x}) \\ &= \beta_0 + \sum_{m' \neq m} \beta_{m'} h_{m'}(\boldsymbol{x}) \; + \; \beta_{m_1} h_{m_1}(\boldsymbol{x}) \; + \; \beta_{m_2} h_{m_2}(\boldsymbol{x}). \end{aligned} \tag{3.16}$$

The optimal split can be determined relative to a loss function, for instance, one can consider the resulting in-sample loss from the Poisson deviance loss. This way we may grow a large complex model (in a stage-wise adaptive manner). By backward pruning (deletion) some of the splits may be deleted again, if they do not contribute sufficiently to the reduction in cross-validation error. For computational reasons, in MARS calibrations one often uses the GCV criterion, which provides a more crude error estimation in terms of an adjusted in-sample error, see (3.13) above and (9.20) in Hastie et al. [62]. We do not provide more details on pruning and deletion here, but pruning in a similar context will be treated in more detail in Chapter 6 in the context of regression trees.
We remark the that refinement (3.16) can also be understood as a boosting step. Boosting is going to be studied in Section 7.4.

Chapter 4

Credibility Theory

In Chapter 2 on GLMs we have seen that many years at risk are needed in order to draw statistically firm conclusions. This holds true, in particular, for problems with low expected frequencies. In this chapter we present credibility methods that allow us to smooth predictors and estimators with other sources of information in situations where we do not have sufficient volume or in situations where we have outliers. Credibility methods are based on Bayesian statistics. We do not present the entire credibility theory framework here, but we only consider some special cases that are related to the Poisson claims frequency model and to a binary classification problem. For a comprehensive treatment of credibility theory we refer to the book of Bühlmann–Gisler [18]. Along the way we also meet the important technique of regularization. Moreover, we discuss simulation methods such as the Markov chain Monte Carlo (MCMC) method to numerically calculate Bayesian posterior distributions.

4.1 The Poisson-gamma model for claims counts

4.1.1 Credibility formula

In Section 1.2 we have assumed that $N \sim \text{Poi}(\lambda v)$ for a fixed expected frequency parameter $\lambda > 0$ and for given years at risk $v > 0$. In this chapter we do not assume that the expected frequency parameter λ is fixed, but it is modeled by a strictly positive random variable Λ. This random variable Λ may have different interpretations: (i) there is some uncertainty involved in the true expected frequency parameter and we reflect this uncertainty by a random variable Λ; (ii) we have a heterogeneous portfolio of different risks and we choose at random a policy from this portfolio. The latter is similar to a heterogeneous situation where a priori all policies are equivalent. Here, we make a specific distributional assumption for Λ by choosing a gamma prior distribution having density for $\theta \in \mathbb{R}_+$

$$\pi(\theta) = \frac{c^\gamma}{\Gamma(\gamma)}\ \theta^{\gamma-1}\ \exp\{-c\theta\},$$

with shape parameter $\gamma > 0$ and scale parameter $c > 0$. Note that the mean and variance of the gamma distribution are given by γ/c and γ/c^2, respectively. For more information on the gamma distribution we refer to Section 3.2.1 in Wüthrich [135].

Definition 4.1 (Poisson-gamma model)**.** *Assume there is a fixed given volume $v > 0$.*

- *Conditionally, given Λ, the claims count $N \sim \mathrm{Poi}(\Lambda v)$.*
- *$\Lambda \sim \Gamma(\gamma, c)$ with prior shape parameter $\gamma > 0$ and prior scale parameter $c > 0$.*

Theorem 4.2. *Assume N follows the Poisson-gamma model of Definition 4.1. The posterior distribution of Λ, conditional on N, is given by*

$$\Lambda|_N \;\sim\; \Gamma\left(\gamma + N, c + v\right).$$

Proof. The Bayes' rule says that the posterior density of Λ, given N, is given by (up to normalizing constants)

$$\pi(\theta|N) \;\propto\; e^{-\theta v}\, \frac{(\theta v)^N}{N!}\, \frac{c^\gamma}{\Gamma(\gamma)}\, \theta^{\gamma-1} e^{-c\theta} \;\propto\; \theta^{\gamma+N-1} e^{-(c+v)\theta}.$$

This is a gamma density with the required properties. □

Remarks 4.3.

- The posterior distribution is again a gamma distribution with updated parameters. The parameter update is given by

$$\gamma \;\mapsto\; \gamma^{\mathrm{post}} = \gamma + N \qquad \text{and} \qquad c \;\mapsto\; c^{\mathrm{post}} = c + v.$$

  Often γ and c are called the *prior parameters* and γ^{post} and c^{post} the *posterior parameters*.

- The remarkable property of the Poisson-gamma model is that the posterior distribution belongs to the same family of distributions as the prior distribution. There are more examples of this kind, many of these examples belong to the EDF with conjugate priors, see Bühlmann–Gisler [18].

- For the estimation of the unknown parameter Λ we obtain the following prior estimator and the corresponding posterior (Bayesian) estimator

$$\begin{aligned} \lambda \;&\overset{\mathrm{def.}}{=}\; \mathbb{E}[\Lambda] \;=\; \frac{\gamma}{c}, \\ \widehat{\lambda}^{\mathrm{post}} \;&\overset{\mathrm{def.}}{=}\; \mathbb{E}[\Lambda|N] \;=\; \frac{\gamma^{\mathrm{post}}}{c^{\mathrm{post}}} \;=\; \frac{\gamma + N}{c + v}. \end{aligned}$$

- Assume that the claims counts $N_1, \ldots, N_n$ are conditionally, given Λ, independent and Poisson distributed with conditional means Λv_i, for $i = 1, \ldots, n$. Lemma 1.3 implies that

$$N = \sum_{i=1}^n N_i|_\Lambda \;\sim\; \mathrm{Poi}\left(\Lambda \sum_{i=1}^n v_i\right).$$

  Thus, we can apply Theorem 4.2 also to the aggregate portfolio $N = \sum_i N_i$, if the claims counts N_i all belong to the *same* frequency parameter Λ.

- The unconditional distribution of N under the Poisson-gamma model is a negative binomial distribution, see Proposition 2.20 in Wüthrich [135].

Corollary 4.4. *Assume N follows the Poisson-gamma model of Definition 4.1. The posterior estimator $\widehat{\lambda}^{\text{post}}$ has the following credibility form*

$$\widehat{\lambda}^{\text{post}} = \alpha \widehat{\lambda} + (1-\alpha)\, \lambda,$$

with credibility weight α and observation based estimator $\widehat{\lambda}$ given by

$$\alpha = \frac{v}{c+v} \in (0,1) \qquad \text{and} \qquad \widehat{\lambda} = \frac{N}{v}.$$

The (conditional) mean square error of this estimator is given by

$$\mathbb{E}\left[\left(\Lambda - \widehat{\lambda}^{\text{post}} \right)^2 \middle| N \right] = \frac{\gamma^{\text{post}}}{(c^{\text{post}})^2} = (1-\alpha)\, \frac{1}{c}\, \widehat{\lambda}^{\text{post}}.$$

Proof. In view of Theorem 4.2 we have for the posterior mean

$$\widehat{\lambda}^{\text{post}} \;=\; \frac{\gamma + N}{c+v} \;=\; \frac{v}{c+v}\, \frac{1}{v}\, N + \frac{c}{c+v}\, \frac{\gamma}{c} \;=\; \alpha\, \widehat{\lambda} + (1-\alpha)\, \lambda.$$

This proves the first claim. For the estimation uncertainty we have

$$\mathbb{E}\left[\left(\Lambda - \widehat{\lambda}^{\text{post}} \right)^2 \middle| N \right] \;=\; \text{Var}\,(\Lambda|\, N) \;=\; \frac{\gamma^{\text{post}}}{(c^{\text{post}})^2} \;=\; (1-\alpha)\, \frac{1}{c}\, \widehat{\lambda}^{\text{post}}.$$

This proves the claim. □

Remarks 4.5.

- Corollary 4.4 shows that the posterior estimator $\widehat{\lambda}^{\text{post}}$ is a credibility weighted average between the prior guess λ and the MLE $\widehat{\lambda}$ with credibility weight $\alpha \in (0,1)$.
- The credibility weight α has the following properties (which balances the weights given in the posterior estimator $\widehat{\lambda}^{\text{post}}$):
  1. for volume $v \to \infty$: $\alpha \to 1$;
  2. for volume $v \to 0$: $\alpha \to 0$;
  3. for prior uncertainty going to infinity, i.e. $c \to 0$: $\alpha \to 1$;
  4. for prior uncertainty going to zero, i.e. $c \to \infty$: $\alpha \to 0$.

  Note that
  $$\text{Var}\,(\Lambda) = \frac{\gamma}{c^2} = \frac{1}{c}\, \lambda.$$
  For c large we have an *informative prior distribution*, for c small we have a *vague prior distribution* and for $c = 0$ we have a *non-informative* or *improper prior distribution*. The latter means that we have no prior parameter knowledge (this needs to be understood in an asymptotic sense). These statements are all done under the assumption that the prior mean $\lambda = \gamma/c$ remains constant.

- In motor insurance it often happens that $N = 0$. In this case, $\widehat{\lambda} = 0$ and we cannot meaningfully calibrate a Poisson model with MLE because we receive a degenerate model. However, the resulting posterior estimator $\widehat{\lambda}^{\text{post}} = (1-\alpha)\lambda > 0$ still leads to a sensible model calibration in the credibility approach. This is going to be important in the application of Poisson regression trees in Chapter 6.

4.1.2 Maximum a posteriori estimator

In the previous section we have derived the posterior estimator $\widehat{\lambda}^{\text{post}}$ for the claims frequency based on observation N. We have calculated the posterior distribution of Λ, given N. Its log-likelihood function is given by

$$\log \pi(\theta|N) \quad \propto \quad (\gamma + N - 1) \log \theta - (c + v)\, \theta.$$

If we maximize this log-likelihood function we receive the maximum a posteriori (MAP) estimator given by

$$\widehat{\lambda}^{\text{MAP}} = \frac{\gamma + N - 1}{c + v} = \widehat{\lambda}^{\text{post}} - \frac{\alpha}{v}. \tag{4.1}$$

Note that the MAP estimator is always positive for $\gamma > 1$, the latter corresponds to a prior coefficient of variation $\gamma^{-1/2}$ being bounded from above by 1. Thus, if we have a prior distribution for Λ which is informative with maximal coefficient of variation of 1, the MAP estimator will always be positive.

4.1.3 Example in motor insurance pricing

The credibility formula of Corollary 4.4 is of particular interest when the portfolio is small, i.e. if v is small, because in that case we receive a credibility weight α that substantially differs from 1. To apply this model we a need prior mean $\lambda = \gamma/c > 0$ and a scale parameter $c > 0$. These parameters may either be chosen from expert opinion or they may be estimated from broader portfolio information. In the present example we assume broader portfolio information and we calibrate the model according to Section 4.10 of Bühlmann–Gisler [18], for full details we refer to this latter reference.

Model Assumptions 4.6. *Assume we have independent portfolios $i = 1, \ldots, n$ all following the Poisson-gamma model of Definition 4.1 with portfolio dependent volumes $v_i > 0$ and portfolio independent prior parameters $\gamma, c > 0$.*

This model is a special case of the Bühlmann–Straub model, see Bühlmann–Gisler [18]. Corollary 4.4 implies that for each portfolio $i = 1, \ldots, n$ we have

$$\widehat{\lambda}_i^{\text{post}} = \alpha_i \widehat{\lambda}_i + (1 - \alpha_i)\, \lambda,$$

with credibility weights α_i and observation based estimators $\widehat{\lambda}_i$ given by

$$\alpha_i = \frac{v_i}{c + v_i} \in (0, 1) \qquad \text{and} \qquad \widehat{\lambda}_i = \frac{N_i}{v_i},$$

and the prior mean is given by $\lambda = \gamma/c$. The optimal estimator (in a credibility sense, homogeneous credibility estimator) of this prior parameter is given by

$$\widehat{\lambda}^{\alpha} = \frac{1}{\sum_{i=1}^n \alpha_i} \sum_{i=1}^n \alpha_i \widehat{\lambda}_i.$$

Thus, there only remains to determine the scale parameter $c > 0$. To estimate this last parameter we apply the iterative procedure given in Bühlmann–Gisler [18], pages 102–103. This procedure provides estimates $\widehat{\tau}^2$ and $\widehat{\lambda}_0$ from which we estimate

$$\widehat{c} = \widehat{\lambda}_0 / \widehat{\tau}^2,$$

for details we refer to Bühlmann–Gisler [18]. This then provides estimated credibility weights

$$\widehat{\alpha}_i = \frac{v_i}{\widehat{c} + v_i} \in (0, 1). \tag{4.2}$$

We apply this credibility model to the car insurance pricing data generated in Appendix A. As portfolios $i = 1, \ldots, n$ we choose the 26 Swiss cantons `ct` and we consider the corresponding volumes v_i and the observed number of claims N_i in each Swiss canton. For the first analysis we consider the whole portfolio which provides a total of $\sum_i v_i = 253'022$ years at risk. The largest Swiss canton `ZH` has $42'644$ years at risk, and the smallest Swiss canton `AI` has $1'194$ years at risk. For the second example we only consider drivers with an age below 20 for which we have a total of $\sum_i v_i = 1'741$ years at risk. The largest Swiss canton `ZH` has 283 years at risk (drivers below age 20), and the smallest Swiss canton `AR` has 4 years at risk in this second example.
If we consider the entire portfolio we obtain fast convergence of the iterative procedure (4 iterations) and we set $\widehat{c} = 456$ and $\widehat{\lambda}^{\alpha} = 10.0768\%$ (based on the entire portfolio). We use this same estimate $\widehat{c} = 456$ for both examples, the entire portfolio and the young drivers only portfolio. From this we can calculate the estimated credibility weights $\widehat{\alpha}_i$ for all Swiss cantons $i = 1, \ldots, n$. The results are presented in Figure 4.1. For the entire

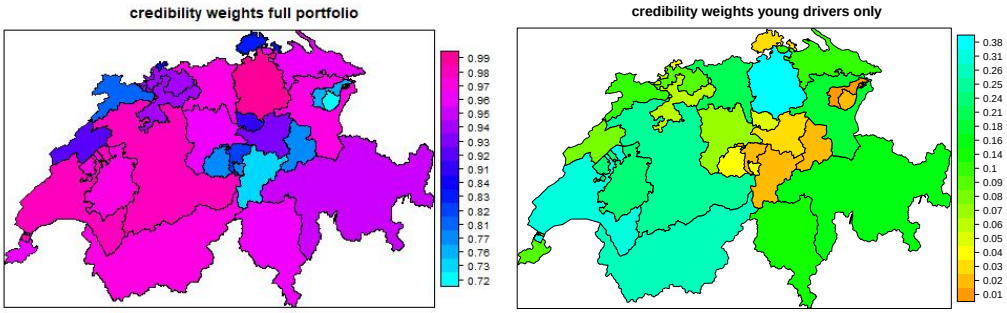


Figure 4.1: Estimated credibility weights (4.2) of (lhs) the entire portfolio and of (rhs) the young drivers only portfolio.

portfolio the estimated credibility weights lie between 72% and 99%. For the young drivers only portfolio the estimated credibility weights are between 0.8% and 40%. Using these credibility weights we could then calculate the Bayesian frequency estimators (which are smoothed versions of the MLE frequency estimators). This finishes our example. ■

4.2 The binomial-beta model for classification

4.2.1 Credibility formula

There is a similar credibility formula for the binary classification problem when we assume that the prior distribution of the parameter is given by a beta distribution.

Definition 4.7 (Binomial-beta model)**.**

- *Conditionally, given* Θ*,* $Y_1, \ldots, Y_n$ *are i.i.d. Bernoulli distributed with success probability* Θ*.*
- $\Theta \sim \text{Beta}(a, b)$ *with prior parameters* $a, b > 0$*.*

Theorem 4.8. *Assume* $Y_1, \ldots, Y_n$ *follow the binomial-beta model of Definition 4.7. The posterior distribution of* Θ*, conditional on* $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$*, is given by*

$$\Theta|_{\boldsymbol{Y}} \;\sim\; \text{Beta}\left(a + \sum_{i=1}^n Y_i, b + n - \sum_{i=1}^n Y_i\right) \;\stackrel{\text{def.}}{=}\; \text{Beta}\left(a^{\text{post}}, b^{\text{post}}\right).$$

Proof. Using the Bayes' rule, the posterior density of Θ is given by (up to normalizing constants)

$$\pi(\theta|\boldsymbol{Y}) \;\propto\; \left(\prod_{i=1}^n \theta^{Y_i}(1-\theta)^{1-Y_i}\right) \theta^{a-1}(1-\theta)^{b-1}.$$

This is a beta density with the required properties. □

For the estimation of the unknown success probability we obtain the following prior and posterior (Bayesian) estimators

$$\begin{aligned} p \;&\stackrel{\text{def.}}{=}\; \mathbb{E}[\Theta] \;=\; \frac{a}{a+b}, \\ \widehat{p}^{\text{post}} \;&\stackrel{\text{def.}}{=}\; \mathbb{E}[\Theta|\boldsymbol{Y}] \;=\; \frac{a^{\text{post}}}{a^{\text{post}} + b^{\text{post}}} \;=\; \frac{a + \sum_{i=1}^n Y_i}{a+b+n}. \end{aligned}$$

Corollary 4.9. *Assume* $Y_1, \ldots, Y_n$ *follow the binomial-beta model of Definition 4.7. The posterior estimator* $\widehat{p}^{\text{post}}$ *has the following credibility form*

$$\widehat{p}^{\text{post}} = \alpha\, \widehat{p} + (1-\alpha)\, p.$$

with credibility weight α *and observation based estimator* $\widehat{p}$ *given by*

$$\alpha = \frac{n}{a+b+n} \in (0,1) \qquad \text{and} \qquad \widehat{p} = \frac{\sum_{i=1}^n Y_i}{n}.$$

The (conditional) mean square error of this estimator is given by

$$\mathbb{E}\left[\left(\Theta - \widehat{p}^{\text{post}}\right)^2 \middle| \boldsymbol{Y}\right] = \frac{1}{1+a+b+n}\, \widehat{p}^{\text{post}}\left(1 - \widehat{p}^{\text{post}}\right).$$

Proof. For the estimation uncertainty we have

$$\mathbb{E}\left[\left(\Theta-\widehat{p}^{\text{post}}\right)^2\Big|\boldsymbol{Y}\right] \;=\; \text{Var}\left(\Theta|\,\boldsymbol{Y}\right) \;=\; \frac{a^{\text{post}}b^{\text{post}}}{(1+a^{\text{post}}+b^{\text{post}})(a^{\text{post}}+b^{\text{post}})^2}.$$

This proves the claim. □

4.2.2 Maximum a posteriori estimator

In the previous section we have derived the posterior estimator $\widehat{p}^{\text{post}}$. The corresponding log-likelihood function was given by

$$\log\pi(\theta|\boldsymbol{Y}) \;\propto\; \left(a+\sum_{i=1}^n Y_i-1\right)\log\theta+\left(b+n-\sum_{i=1}^n Y_i-1\right)\log(1-\theta).$$

If we maximize this log-likelihood function we receive the MAP estimator

$$\widehat{p}^{\text{MAP}}=\frac{a+\sum_{i=1}^n Y_i-1}{a+b+n-2}=\frac{a+b+n}{a+b+n-2}\left(\widehat{p}^{\text{post}}-\frac{\alpha}{n}\right).$$

Here, we require that $a>1$ for having a MAP estimator that has a positive probability for any observation $\boldsymbol{Y}$.

4.3 Regularization and Bayesian MAP estimators

4.3.1 Bayesian posterior parameter estimator

For illustrative purposes we reconsider the GLM of Chapter 2. However, the theory presented in this section applies to any other parametric regression model. We consider GLM (2.4) for independent cases $(N_i,\boldsymbol{x}_i,v_i)\in\mathcal{D}$. The joint density of these cases is for regression parameter $\boldsymbol{\beta}$ given by

$$f_{\boldsymbol{\beta}}(N_1,\ldots,N_n)=\exp\left\{\ell_{\boldsymbol{N}}(\boldsymbol{\beta})\right\}=\prod_{i=1}^n e^{-\exp\langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle v_i}\;\frac{(\exp\langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle v_i)^{N_i}}{N_i!}.$$

Bayesian modeling in this context means that we choose a prior density $\pi(\boldsymbol{\beta})$ for the regression parameter $\boldsymbol{\beta}$. The joint distribution of the cases and the regression parameter then reads as

$$f(N_1,\ldots,N_n,\boldsymbol{\beta})=\prod_{i=1}^n e^{-\exp\langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle v_i}\;\frac{(\exp\langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle v_i)^{N_i}}{N_i!}\;\pi(\boldsymbol{\beta}). \tag{4.3}$$

After having observed data $\mathcal{D}$, the parameter vector $\boldsymbol{\beta}$ has posterior density

$$\pi\left(\boldsymbol{\beta}|\,\mathcal{D}\right)\;\propto\; f(N_1,\ldots,N_n,\boldsymbol{\beta}), \tag{4.4}$$

where we have dropped the normalizing constants. Thus, we receive an explicit functional form for the posterior density of the parameter vector $\boldsymbol{\beta}$, conditionally given the data $\mathcal{D}$. Simulation methods like MCMC algorithms, see Section 4.4, or sequential Monte Carlo (SMC) samplers then allow us to evaluate numerically the Bayesian (credibility) estimator

$$\widehat{\boldsymbol{\beta}}^{\text{post}}=\mathbb{E}\left[\boldsymbol{\beta}|\,\mathcal{D}\right]=\int\boldsymbol{\beta}\;\pi\left(\boldsymbol{\beta}|\,\mathcal{D}\right)d\boldsymbol{\beta}. \tag{4.5}$$

This Bayesian estimator considers both, the prior information $\pi(\boldsymbol{\beta})$ and the data $\mathcal{D}$.

4.3.2 Ridge and LASSO regularization

The joint distribution of the data and the regression parameter (4.3) can also be viewed as a regularized version of the log-likelihood function $\ell_{\boldsymbol{N}}(\boldsymbol{\beta})$, where the prior density π is used for regularization. We have already met this idea in the calibration of GAMs, see (3.7). Assume that $\pi(\boldsymbol{\beta}) \propto \exp\{-\eta\|\boldsymbol{\beta}\|_p^p\}$ for some tuning parameter $\eta > 0$ and $\|\cdot\|_p$ denoting the L^p-norm for some $p \geq 1$. In this case we receive joint log-likelihood function of data and regression parameter

$$\ell_{\boldsymbol{N}}(\boldsymbol{\beta}) + \log \pi(\boldsymbol{\beta}) \;\propto\; \sum_{i=1}^{n} -\exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle v_i + N_i \left(\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle + \log v_i\right) - \eta\|\boldsymbol{\beta}\|_p^p. \tag{4.6}$$

If we now calculate the MAP estimator $\widehat{\boldsymbol{\beta}}^{\text{MAP}}$ we receive an L^p-regularized version of the MLE $\widehat{\boldsymbol{\beta}}$ given in Proposition 2.2. Maximization of objective function (4.6) implies that too large values for $\boldsymbol{\beta}$ are punished (regularized/shrunk towards zero) with regularization parameter (tuning parameter) $\eta > 0$. In particular, this helps us to control complex models to not over-fit to the data (in-sample).

Remarks 4.10 (ridge and LASSO regularization)**.**

- The last term in (4.6) tries to keep $\boldsymbol{\beta}$ small. Typically, the intercept β_0 is excluded from this regularization because otherwise regularization induces a bias towards 0, see *balance property* of Proposition 2.4.
- For a successful application of regularization one requires that all feature components live on the same scale. This may require scaling using, for instance, the MinMaxScaler, see (5.14), below. Categorical feature components may require a group treatment, see Section 4.3 in Hastie et al. [63].
- $p = 2$ gives a Gaussian prior distribution, and we receive the so-called *ridge regularization* or Tikhonov regularization [127]. Note that we can easily generalize this ridge regularization to any Gaussian distribution. Assume we have a prior mean $\boldsymbol{b}$ and a positive definite prior covariance matrix Σ, then we can consider component-specific regularization
$$\log \pi(\boldsymbol{\beta}) \;\propto\; -\frac{\eta}{2}\left(\boldsymbol{\beta} - \boldsymbol{b}\right)' \Sigma^{-1} \left(\boldsymbol{\beta} - \boldsymbol{b}\right).$$
- $p = 1$ gives a Laplace prior distribution, and we receive the so-called *LASSO regularization* (least absolute shrinkage and selection operator), see Tibshirani [126]. LASSO regularization shrinks (unimportant) components to exactly zero, i.e. LASSO regularization can be used for variable selection.
- Optimal tuning parameters η are determined by cross-validation.

Example 4.11 (regularization of GLM1)**.** We revisit Example 2.10 (GLM1) and we apply ridge and LASSO regularization according to (4.6). We consider the feature components `age` and `ac` using the categorical classes as defined in Figure 2.2.
The R code is provide in Listing 4.1. On line 1 we define the design matrix only consisting of the categorized '`age` class' and '`ac` class', respectively. On lines 2-3 we fit

Listing 4.1: R code `glmnet` for regularization

```
X <- model.matrix(~ ageGLM + acGLM, dat)
glm.ridge <- glmnet(x=X,y=dat$claims,family="poisson",alpha=0,offset=log(dat$expo))
exp(predict(glm.ridge, newx=X, newoffset=log(dat$expo)))
```

this regularized regression model. Parameter `alpha` $= 0$ gives ridge regularization, and `alpha` $= 1$ gives LASSO regularization. Running this algorithm as in Listing 4.1 fits the regularized GLM for 100 different values of the tuning parameter η. These tuning parameters are in $[0.00137, 13.72341]$ for ridge regularization and in $[0.00004, 0.01372]$ for LASSO regularization in our example. Moreover, the intercept β_0 is excluded from regularization.

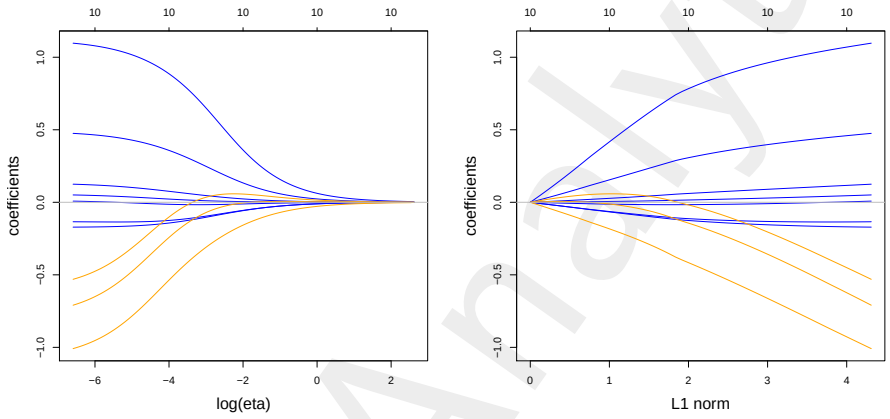


Figure 4.2: Estimated regression parameters $\widehat{\boldsymbol{\beta}}^{\mathrm{MAP}}$ under ridge regularization, the x-axis shows (lhs) $\log(\eta)$ and (rhs) the total L^1-norm of the regression parameter (excluding the intercept), in blue color are the parameters for the '`age` classes' and in orange color the ones for the '`ac` classes', see also Listing 2.2.

In Figure 4.2 we illustrate the resulting ridge regularized regression parameters $\widehat{\boldsymbol{\beta}}^{\mathrm{MAP}}$ for different tuning parameters η (excluding the intercept). We observe that the components in $\widehat{\boldsymbol{\beta}}^{\mathrm{MAP}}$ shrink with increasing tuning parameter towards the homogeneous model.
In Table 4.1 lines (Ch4.1)-(Ch4.3) we present the resulting errors for different tuning parameters. We observe that choosing different tuning parameters $\eta \geq 0$ continuously closes the gap between the homogeneous model and model GLM1.
In Figure 4.3 we illustrate the resulting LASSO regularized regression parameters $\widehat{\boldsymbol{\beta}}^{\mathrm{MAP}}$ for different tuning parameters η. We see that this is different from ridge regularization because parameters are set/shrunk to exactly zero for increasing tuning parameter η. The x-axis on the top of Figure 4.3 shows the number of non-zero components in $\widehat{\boldsymbol{\beta}}^{\mathrm{MAP}}$ (excluding the intercept). We note that at $\eta = 0.0001$ the regression parameter for `age31to40` set to zero and at $\eta = 0.0009$ the one for `age41to50` is set to zero. Since `age51to60` is the reference label, this LASSO regularization implies that we receive one large `age` class from age 31 to age 60 for $\eta = 0.0009$. Lines (Ch4.4)-(Ch4.6) of Table

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch4.1) ridge $\eta = 13.723$	13s†	11			1.3439	29.1065	10.2691%
(Ch4.2) ridge $\eta = 0.1438$	13s†	11			1.0950	28.8504	10.2691%
(Ch4.3) ridge $\eta = 0.0014$	13s†	11			0.6091	28.3547	10.2691%
(Ch4.4) LASSO $\eta = 0.0137$	7s†	1			1.3439	29.1065	10.2691%
(Ch4.5) LASSO $\eta = 0.0009$	7s†	9			0.6329	28.3791	10.2691%
(Ch4.6) LASSO $\eta = 0.0001$	7s†	11			0.6053	28.3511	10.2691%
(Ch2.1) GLM1	3.1s	11	28.3543	28.3544	0.6052	28.3510	10.2691%

Table 4.1: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, the run time† is received by simultaneously fitting the model for 100 tuning parameters η, and '# param.' gives the number of estimated model parameters, this table follows up from Table 2.4.

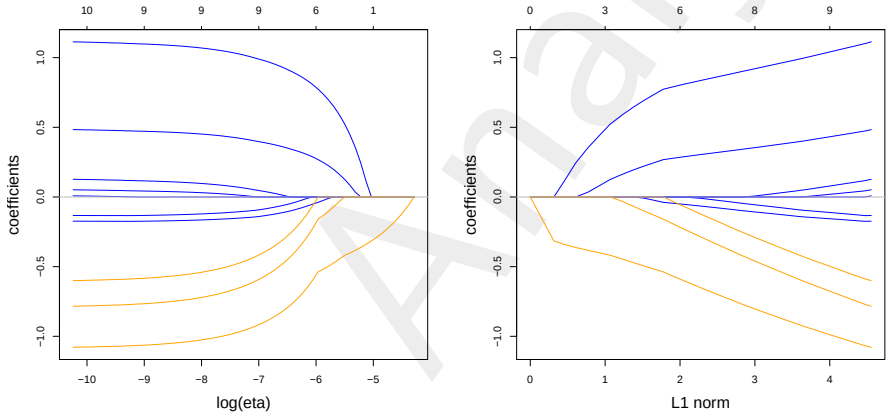


Figure 4.3: Estimated regression parameters $\widehat{\boldsymbol{\beta}}^{\text{MAP}}$ under LASSO regularization, the x-axis shows (lhs) $\log(\eta)$ and (rhs) the total L^1-norm of the regression parameter (excluding the intercept), in blue color are the parameters for the '`age` classes' and in orange color the ones for the '`ac` classes', see also Listing 2.2.

4.1 show that also LASSO regularization closes the gap between the homogeneous model and model GLM1. This finishes this example. ■

In the previous example we have seen that the ridge and LASSO regressions behave fundamentally differently, namely, LASSO regression shrinks certain parameters perfectly to zero whereas ridge regression does not. We briefly analyze the reason therefore, and we give some more insight into regularization. For references we refer to Hastie et al. [63], Chapter 16 in Efron–Hastie [37], and Chapter 6 in Wüthrich–Merz [141]. We start from the following optimization problem

$$\arg\min_{\boldsymbol{\beta}} \; -\ell_{\boldsymbol{N}}(\boldsymbol{\beta}), \qquad \text{subject to } \|\boldsymbol{\beta}\|_p^p \leq t, \tag{4.7}$$

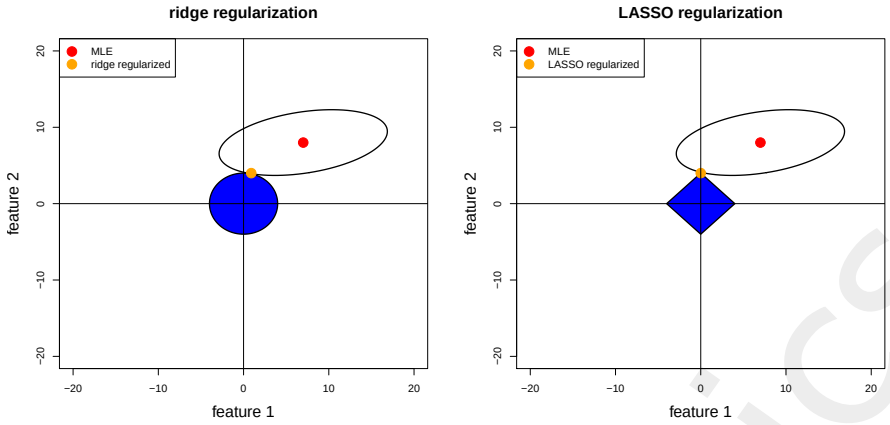


Figure 4.4: Illustration of optimization problem (4.7) under a budget constraint (lhs) for $p = 2$ and (rhs) $p = 1$; this figure is taken from Wüthrich–Merz [141].

for some given budget constraint $t \in \mathbb{R}_+$ and $p \geq 1$. This is a convex optimization problem with a convex constraint. Version (4.6) is obtained as the Lagrange version of this optimization problem (4.7) with fixed Lagrange multiplier η. We illustrate optimization problem (4.7) in Figure 4.4. The red dot shows the MLE that maximizes the unconstrained log-likelihood $\ell_{\boldsymbol{N}}(\boldsymbol{\beta})$. The convex curve around this MLE shows a level set of the log-likelihood function given by $\{\boldsymbol{\beta}; \ell_{\boldsymbol{N}}(\boldsymbol{\beta}) = c_0\}$ for a given level c_0. The blue area shows the constraint in (4.7), the (lhs) shows a Euclidean ball for $p = 2$ and the (rhs) shows a square for $p = 1$. The optimal regularized estimate for $\boldsymbol{\beta}$ is obtained by the tangential intersection of the appropriate level set with the blue budget constraint. This is illustrated by the orange dots. We observe that for $p = 1$ this might be in the corner of the square (Figure 4.4, rhs), this is the situation where the parameter for the first feature component shrinks exactly to zero. For $p = 2$ this does not happen, a.s.

4.4 Markov chain Monte Carlo method

In this section we give the basics of Markov chain Monte Carlo (MCMC) simulation methods. We refrain from giving rigorous derivations and proofs but we refer to the relevant MCMC literature like Congdon [28], Gilks et al. [53], Green [57, 58], Johansen et al. [75], Neal [97] or Robert [113].

A main problem in Bayesian modeling is that posterior distributions of type (4.4) cannot be found explicitly, i.e. we can neither directly simulate from that posterior distribution nor can we calculate the (posterior) Bayesian credibility estimator (4.5) explicitly. In this section we provide an algorithm that helps us to approximate posterior distributions. For this discussion we switchback the notation from $\boldsymbol{\beta}$ to Θ because the latter is more standard in a Bayesian modeling context.

In a nutshell, if a probability density $\pi(\theta)$ (or $\pi(\theta|\mathcal{D})$ in a Bayesian context) of a random variable Θ is given up to its normalizing constant (i.e. if we explicitly know its functional form in θ), then we can design a discrete time Markov process that converges (in distribu-

tion) to π (or $\pi(\cdot|\mathcal{D})$, respectively). More precisely, we can design a (time inhomogeneous) Markov process that converges (under reversibility, irreducibility and aperiodicity) to its (unique) stationary distribution (also called equilibrium distribution).
Assume for simplicity that π is discrete. If the (irreducible and aperiodic) Markov chain $(\Theta^{(t)})_{t\geq 0}$ has the following invariance property for π

$$\pi(\theta_i) = \sum_{\theta_j} \pi(\theta_j)\ \mathbb{P}\left[\Theta^{(t+1)} = \theta_i \middle| \Theta^{(t)} = \theta_j\right], \qquad \text{for all } \theta_i \text{ and } t \geq 0, \tag{4.8}$$

then π is the (unique) stationary distribution of $(\Theta^{(t)})_{t\geq 0}$. Based on the latter property, we can use the samples $(\Theta^{(t)})_{t>t_0}$ as an empirical approximation to π after the Markov chain has sufficiently converged, i.e. for sufficiently large t_0. This t_0 is sometimes also called the burn-in period.
Thus, we are aiming at constructing an irreducible and aperiodic Markov chain that satisfies invariance property (4.8) for π. A Markov chain is said to fulfill the detailed balance condition (is reversible) w.r.t. π if the following holds for all θ_i, θ_j and $t \geq 0$

$$\pi(\theta_i)\ \mathbb{P}\left[\Theta^{(t+1)} = \theta_j \middle| \Theta^{(t)} = \theta_i\right] = \pi(\theta_j)\ \mathbb{P}\left[\Theta^{(t+1)} = \theta_i \middle| \Theta^{(t)} = \theta_j\right]. \tag{4.9}$$

We observe that the detailed balance condition (4.9) implies the invariance property (4.8) for π (sum both sides over θ_j). Therefore, reversibility is sufficient to obtain the invariance property. The aim now is to design (irreducible and aperiodic) Markov chains that are reversible for π; the limiting samples of these Markov chains can be used as empirical approximations to π. Crucial for this design is that we only need to know the functional form of the density π; note that the normalizing constant of π cancels in (4.9).

4.4.1 Metropolis–Hastings algorithm

The goal is to design an irreducible and aperiodic Markov chain that fulfills the detailed balance condition (4.9) for density π. The main idea of this design goes back to Metropolis and Hastings [95, 64] and uses an acceptance-rejection method.
Assume that the Markov chain is in state $\Theta^{(t)}$ at algorithmic time t and we would like to simulate the next state $\Theta^{(t+1)}$ of the chain. For the simulation of this next state we choose a proposal density $q(\cdot|\Theta^{(t)})$ that may depend on the current state $\Theta^{(t)}$ (and also on the algorithmic time t, not indicated in the notation).

Metropolis–Hastings Algorithm.

---

1. In a first step we simulate a proposed next state

$$\Theta^* \ \sim\ q(\cdot|\Theta^{(t)}).$$

2. Using this proposed state we calculate the acceptance probability

$$\alpha(\Theta^{(t)}, \Theta^*) = \min\left\{\frac{\pi(\Theta^*)q(\Theta^{(t)}|\Theta^*)}{\pi(\Theta^{(t)})q(\Theta^*|\Theta^{(t)})}, 1\right\}. \tag{4.10}$$

3. Simulate independently $U \sim \text{Uniform}[0,1]$.

4. Set for the state at algorithmic time $t+1$

$$\Theta^{(t+1)} = \begin{cases} \Theta^* & \text{if } U \leq \alpha(\Theta^{(t)}, \Theta^*), \\ \Theta^{(t)} & \text{otherwise.} \end{cases}$$

Lemma 4.12. *This algorithm satisfies the detailed balance condition* (4.9) *for* π*, and hence is invariant according to* (4.8) *for* π*.*

This lemma is proved in Section 4.5, below.

Thus, the Metropolis–Hastings (MH) algorithm provides a reversible Markov chain that converges to the stationary distribution π (under irreducibility and aperiodicity). The crucial point why this algorithm works for densities π that are only known up to their normalizing constants lies in the definition of the acceptance probability (4.10); note that the acceptance probability is the only place where π is used. In the definition of the acceptance probability (4.10) we consider the ratio $\pi(\Theta^*)/\pi(\Theta^{(t)})$ and therefore the normalizing constant cancels.

The remaining freedom is the choice of the (time inhomogeneous) proposal densities q. In general, we aim at using a choice that leads to a fast convergence to the stationary distribution π. However, such a "good" choice is not straightforward. In a Gaussian context it was proved that the optimal average acceptance rate of the proposal in the MH algorithm is 0.234, see Roberts et al. [114].

A special case is the so-called random walk Metropolis–Hastings (RW-MH) algorithm (also called Metropolis algorithm). It is obtained by choosing a symmetric proposal density, i.e. $q(\theta_i|\theta_j) = q(\theta_j|\theta_i)$. In this case the acceptance probability simplifies to

$$\alpha(\Theta^{(t)}, \Theta^*) = \min\left\{\frac{\pi(\Theta^*)}{\pi(\Theta^{(t)})}, 1\right\}.$$

If Θ is not low dimensional, then often a block update MH algorithm is used. Assume that $\Theta = (\Theta_1, \ldots, \Theta_K)$ can be represented by K components. In that case a MH algorithm can be applied iteratively to each component separately, and we still obtain convergence of the Markov chain to its stationary distribution π. Assume we are in state $\Theta^{(t)}$ at algorithmic time t and we would like to simulate the next state $\Theta^{(t+1)}$. The block update MH algorithm works as follows:

Block Update Metropolis–Hastings Algorithm.

---

1. Initialize the block update $\Theta^{(t+1)} = \Theta^{(t)}$.

2. Iterate for components $k = 1, \ldots, K$

   (a) simulate a proposed state $\Theta_k^* \sim q_k(\cdot|\Theta^{(t+1)})$ for component Θ_k of Θ, and set

$$\Theta^* = (\Theta_1^{(t+1)}, \ldots, \Theta_{k-1}^{(t+1)}, \Theta_k^*, \Theta_{k+1}^{(t+1)}, \ldots, \Theta_K^{(t+1)}); \tag{4.11}$$

(b) calculate the acceptance probability of this newly proposed component

$$\alpha(\Theta^{(t+1)}, \Theta^*) = \min\left\{ \frac{\pi(\Theta^*) q_k(\Theta_k^{(t+1)}|\Theta^*)}{\pi(\Theta^{(t+1)}) q_k(\Theta_k^*|\Theta^{(t+1)})}, 1 \right\};$$

(c) simulate independently $U \sim \text{Uniform}[0,1]$;

(d) set for the component k at algorithmic time $t+1$

$$\Theta_k^{(t+1)} = \begin{cases} \Theta_k^* & \text{if } U \leq \alpha(\Theta^{(t+1)}, \Theta^*), \\ \Theta_k^{(t)} & \text{otherwise.} \end{cases}$$

Observe that this block update MH algorithm updates component by component of $\Theta^{(t)}$ to obtain $\Theta^{(t+1)}$. The (notation in the) initialization is (only) used to simplify the description of the block update, i.e. in fact the initialization does not update any component. This is reflected in (4.11) by the fact that components 1 to $k-1$ are already updated, component k is about to get updated, and components $k+1$ to K are not updated yet. This block update MH algorithm is of special interest for Gibbs sampling, which is described next.

4.4.2 Gibbs sampling

The block update MH algorithm is of special interest if it is possible to perform conditional sampling. Assume that for all $k = 1, \ldots, K$ the density π is so nice that we can directly sample from the conditional distributions

$$\Theta_k|_{(\Theta_1,\ldots,\Theta_{k-1},\Theta_{k+1},\ldots,\Theta_K)} \sim \pi(\cdot|\Theta_1, \ldots, \Theta_{k-1}, \Theta_{k+1}, \ldots, \Theta_K).$$

In this case we may choose as proposal distribution for Θ_k^* in the above block update MH algorithm (called Gibbs sampling in this case)

$$\Theta_k^* \sim q_k(\cdot|\Theta^{(t+1)}) \overset{(d)}{=} \pi(\cdot|\Theta_1^{(t+1)}, \ldots, \Theta_{k-1}^{(t+1)}, \Theta_{k+1}^{(t+1)}, \ldots, \Theta_K^{(t+1)}) \propto \pi(\Theta^*),$$

where we use notation (4.11). These proposals are always accepted because $\alpha \equiv 1$.

4.4.3 Hybrid Monte Carlo algorithm

The hybrid Monte Carlo (HMC) algorithm goes back to Duane et al. [34] and is well described in Neal [97]. Note that the abbreviation HMC is also used for Hamiltonian Monte Carlo because the subsequent Markov process dynamics are based on Hamiltonians. Represent the density π as follows

$$\pi(\theta) = \exp\{\log \pi(\theta)\} \propto \exp\{\ell(\theta)\}.$$

Observe that ℓ is the log-likelihood function of π, (potentially) up to a normalizing constant. The acceptance probability (4.10) in the MH algorithm then reads as

$$\alpha(\Theta^{(t)}, \Theta^*) = \exp\left(\min\left\{\ell(\Theta^*) - \ell(\Theta^{(t)}) + \log q(\Theta^{(t)}|\Theta^*) - \log q(\Theta^*|\Theta^{(t)}), 0\right\}\right).$$

In physics $-\ell(\theta)$ can be interpreted as the potential energy of the system in state θ. We enlarge this system by also accounting for its momentum. In physics this is done by considering the kinetic energy (and using the law of conservation of energy). Assume that Θ and Ψ are independent with joint density

$$f(\theta,\psi) \;=\; \pi(\theta)g(\psi) \;\propto\; \exp\left\{\ell(\theta)-k(\psi)\right\}.$$

We assume that $-\ell(\cdot)$ and $k(\cdot)$ are differentiable, playing the role of potential and kinetic energy, respectively, and we define the Hamiltonian

$$H(\theta,\psi) = -\ell(\theta)+k(\psi).$$

Assume that the dimension of $\theta=(\theta_1,\ldots,\theta_r)$ is $r\in\mathbb{N}$, then a typical choice for $\psi=(\psi_1,\ldots,\psi_r)$ and its kinetic energy is

$$k(\psi) \;=\; \sum_{l=1}^r \frac{\psi_l^2}{2\tau_l^2} \;=\; \frac{1}{2}\psi'\left(\operatorname{diag}(\tau_1^2,\ldots,\tau_r^2)\right)^{-1}\psi,$$

for given parameters $\tau_l>0$. In other words, $g(\psi)$ is a Gaussian density with independent components $\Psi_1,\ldots,\Psi_r$ and covariance matrix $T=\operatorname{diag}(\tau_1^2,\ldots,\tau_r^2)$. The goal is to design a reversible, irreducible and aperiodic Markov process that has f as stationary distribution (note that we silently change to a continuous space and time setting, for instance, for derivatives being well-defined).

The crucial point now is the construction of the Markov process dynamics. Assume that we are in state $(\Theta^{(t)},\Psi^{(t)})$ at time t and we would like to study an infinitesimal time interval dt. For this we consider a change (in infinitesimal time) that leaves the total energy in the Hamiltonian unchanged (law of conservation of energy). Thus, for all components $l=1,\ldots,r$ we choose the dynamics at time t

$$\frac{d\Theta_l^{(t)}}{dt} = \frac{\Psi_l^{(t)}}{\tau_l^2} \qquad \text{and} \qquad \frac{d\Psi_l^{(t)}}{dt} = \left.\frac{\partial \ell(\theta)}{\partial\theta_l}\right|_{\Theta^{(t)}}. \tag{4.12}$$

This implies for the change in the Hamiltonian at time t

$$\begin{aligned}\frac{dH(\Theta^{(t)},\Psi^{(t)})}{dt} &= \sum_{l=1}^r\left[\left.\frac{\partial H(\theta,\psi)}{\partial\theta_l}\right|_{(\Theta^{(t)},\Psi^{(t)})}\frac{d\Theta_l^{(t)}}{dt}+\left.\frac{\partial H(\theta,\psi)}{\partial\psi_l}\right|_{(\Theta^{(t)},\Psi^{(t)})}\frac{d\Psi_l^{(t)}}{dt}\right]\\ &= \sum_{l=1}^r\left[-\left.\frac{\partial \ell(\theta)}{\partial\theta_l}\right|_{\Theta^{(t)}}\frac{d\Theta_l^{(t)}}{dt}+\left.\frac{\partial k(\psi)}{\partial\psi_l}\right|_{\Psi^{(t)}}\frac{d\Psi_l^{(t)}}{dt}\right] \;=\; 0.\end{aligned}$$

Thus, a Markov process $(\Theta^{(t)},\Psi^{(t)})_{t\geq 0}$ which moves according to (4.12) leaves the total energy in the Hamiltonian unchanged. Moreover, this dynamics preserves the volumes, see (3.8) in Neal [97], and it is reversible w.r.t. $f(\theta,\psi)$. This implies that $f(\theta,\psi)$ is an invariant distribution for this (continuous-time) Markov process $(\Theta^{(t)},\Psi^{(t)})_{t\geq 0}$. Note, however, that this Markov process is not irreducible because it cannot leave a given total energy state (and hence cannot explore the entire state space) for a given starting value. This deficiency can be corrected by randomizing the total energy level. Furthermore, for simulation we also need to discretize time in (4.12). These two points are described next.

First, we mention that we can apply Gibbs sampling for the update of the kinetic energy state $\Psi^{(t)}$, given $\Theta^{(t)}$, simply by using the Gaussian density $g(\psi)$. Assume we are in position $(\Theta^{(t)}, \Psi^{(t)})$ at algorithmic time $t \in \mathbb{N}_0$ and we aim at simulating state $(\Theta^{(t+1)}, \Psi^{(t+1)})$. We do this in two steps, first we move the kinetic energy level and then we move along the corresponding (discretized) constant Hamiltonian level.

HYBRID (HAMILTONIAN) MONTE CARLO ALGORITHM.

---

1. Using Gibbs sampling, we set

$$\widehat{\Psi}^{(t+1)}\Big|_{\Theta^{(t)}} \sim \mathcal{N}\left(0, T = \operatorname{diag}(\tau_1^2, \ldots, \tau_r^2)\right).$$

2. For fixed step size $\varepsilon > 0$ and number of steps $L \in \mathbb{N}$, we initialize $\theta^*(0) = \Theta^{(t)}$ and $\psi^*(0) = \widehat{\Psi}^{(t+1)}$, and we consider the leapfrog updates for $j = 1, \ldots, L$

$$\begin{aligned} \psi^*(j-1/2) &= \psi^*(j-1) + \frac{\varepsilon}{2} \nabla_\theta \ell(\theta^*(j-1)), \\ \theta^*(j) &= \theta^*(j-1) + \varepsilon\, T^{-1} \psi^*(j-1/2), \\ \psi^*(j) &= \psi^*(j-1/2) + \frac{\varepsilon}{2} \nabla_\theta \ell(\theta^*(j)). \end{aligned}$$

This provides a proposal for the next state

$$(\Theta^*, \Psi^*) = (\theta^*(L), -\psi^*(L)).$$

Accept this proposal with acceptance probability

$$\alpha\left((\Theta^{(t)}, \widehat{\Psi}^{(t+1)}), (\Theta^*, \Psi^*)\right) = \exp\left(\min\left\{-H(\Theta^*, \Psi^*) + H(\Theta^{(t)}, \widehat{\Psi}^{(t+1)}), 0\right\}\right), \tag{4.13}$$

otherwise keep $(\Theta^{(t)}, \widehat{\Psi}^{(t+1)})$.

We give some remarks. Observe that in step 2 we may update both components $\Theta^{(t)}$ and $\widehat{\Psi}^{(t+1)}$. If the leapfrog discretization were exact (i.e. no discretization error), then it would define, as in (4.12), a reversible Markov process that leaves the total energy and the total volume unchanged, see Section 3.1.3 in Neal [97]. This would imply that the acceptance probability in (4.13) is equal to 1. The MH step with acceptance probability (4.13) corrects for the discretization error (corrects for a potential bias). Moreover, since the kinetic energy $k(\psi)$ is symmetric with respect to zero, the reflection of the momentum in the proposal Ψ^* would not be necessary (if we perform Gibbs sampling in step 1).
Note that the wanted distribution π is obtained by considering the (for the first component) marginalized Markov process $(\Theta^{(t)}, \Psi^{(t)})_{t \geq 0}$ asymptotically.

4.4.4 Metropolis-adjusted Langevin algorithm

The above HMC algorithm provides as special case for $L = 1$ the Metropolis-adjusted Langevin algorithm (MALA), studied in Roberts–Rosenthal [115].

First, observe that the Gibbs sampling step of $\widehat{\Psi}^{(t+1)}$ is independent of everything else, and we can interpret this Gibbs sampling step as a proposal distribution for updating $\Theta^{(t)}$, in particular, because we are only interested into the marginalized process $(\Theta^{(t)})_{t\geq 0}$ of $(\Theta^{(t)}, \Psi^{(t)})_{t\geq 0}$. The leapfrog update for $L = 1$ reads as

$$\begin{aligned}\Theta^* &= \Theta^{(t)} + \frac{\varepsilon^2}{2}\, T^{-1}\nabla_\theta \log \pi(\Theta^{(t)}) + \varepsilon\, T^{-1}\widehat{\Psi}^{(t+1)},\\ \Psi^* &= -\widehat{\Psi}^{(t+1)} - \frac{\varepsilon}{2}\,\nabla_\theta \log \pi(\Theta^{(t)}) - \frac{\varepsilon}{2}\,\nabla_\theta \log \pi(\Theta^*).\end{aligned}$$

From the first line we see that the proposal is chosen as

$$\Theta^*|_{\Theta^{(t)}} \;\sim\; \mathcal{N}\left(\Theta^{(t)} + \frac{\varepsilon^2}{2}T^{-1}\nabla_\theta \log \pi(\Theta^{(t)}), \varepsilon^2 T^{-1}\right). \tag{4.14}$$

Lemma 4.13. *The acceptance probability obtained from* (4.13) *for* $L = 1$ *is equal to the acceptance probability of the MH algorithm using proposal* (4.14).

This lemma is proved in Section 4.5, below.

Remarks and interpretation. The motivation for the study of the MALA has been that the classical MH algorithm may have a poor performance in higher dimensions, because the Markov process may behave like a random walk that explores the space in a rather inefficient way. The idea is to endow the process with an appropriate drift so that it finds the relevant corners of the space more easily. For this purpose the Langevin diffusion is studied: assume that $(W_t)_{t\geq 0}$ is an r-dimensional Brownian motion on our probability space, then we can consider the r-dimensional stochastic differential equation

$$d\Theta^{(t)} \;=\; \frac{1}{2}\nabla_\theta \log \pi(\Theta^{(t)})dt + dW_t.$$

When π is sufficiently smooth, then π is a stationary distribution of this Langevin diffusion. The MALA described by the proposals (4.14) is then directly obtained by a discretized version of this Langevin diffusion for $T = \mathbb{1}$ and for ε describing the step size. The acceptance-rejection step then ensures convergence to the appropriate stationary distribution π. In Roberts–Rosenthal [115] it was proved that (under certain assumptions) the optimal average acceptance rate of the MALA is 0.574. Note that this is vastly different from the 0.234 in the classical MH algorithm, see Roberts et al. [114]. In Neal [97] it is argued that also the MALA may have a too random walk-like behavior and the full power of HMC algorithms is only experienced for bigger L's reflecting bigger steps along the (deterministic) Hamiltonian dynamics.

4.4.5 Example in Markov chain Monte Carlo simulation

In this section we make a simple example for the density π that allows us to empirically compare the performance of the MH algorithm, the MALA and the HMC algorithm. We choose dimension $r = 10$ (which is not large) and let $\Sigma \in \mathbb{R}^{r\times r}$ be a positive definite covariance matrix, the explicit choice is provided in Figure 4.5. We then assume that

$$\pi \;\overset{(\mathrm{d})}{=}\; \mathcal{N}(0, \Sigma).$$

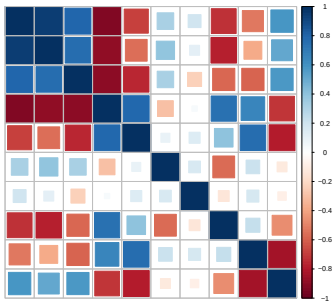


Figure 4.5: Choice of covariance matrix Σ.

Thus, π is a centered multivariate Gaussian density. Of course, in this case we could directly simulate from π. Nevertheless, we will use the MCMC algorithms to approximate samples of π. This allows us to study the performance of these algorithms.

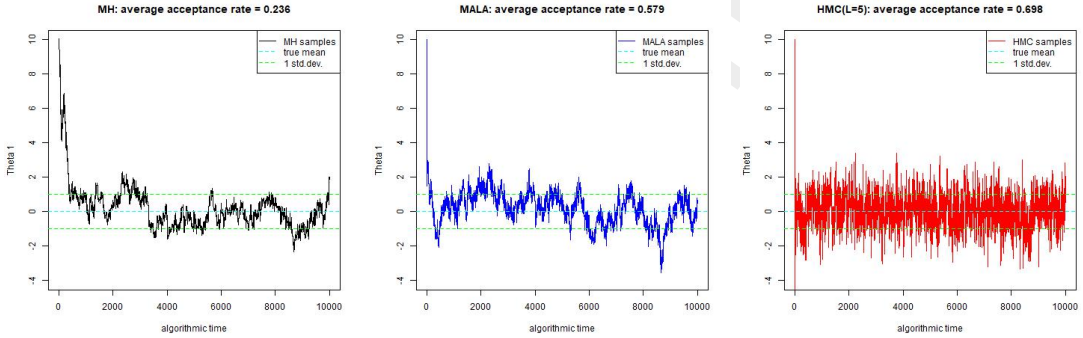


Figure 4.6: Trace plots of the first component $\Theta_1^{(t)}$ of $(\Theta^{(t)})_{t=0,\dots,10000}$ of (lhs, black) the MH algorithm, (middle, blue) the MALA, and (rhs, red) the HMC algorithm with $L = 5$.

We start with the MH algorithm. For the MH algorithm we choose proposal distribution

$$\Theta^* \sim q(\cdot|\Theta^{(t)}) \overset{(\mathrm{d})}{=} \mathcal{N}(\Theta^{(t)}, \varepsilon^2 \mathbb{1}).$$

Thus, we do a RW-MH choice and the resulting acceptance probability is given by

$$\alpha\left(\Theta^{(t)}, \Theta^*\right) = \exp\left(\min\left\{-\frac{1}{2}(\Theta^*)'\Sigma^{-1}\Theta^* + \frac{1}{2}(\Theta^{(t)})'\Sigma^{-1}\Theta^{(t)}, 0\right\}\right).$$

The parameter $\varepsilon > 0$ is fine-tuned so that we obtain an average acceptance rate of roughly 0.234. The trace plot of the first component $(\Theta_1^{(t)})_t$ is given in Figure 4.6 (lhs, black). Note that we choose starting point $\Theta^{(0)} = (10, \dots, 10)'$ and the burn-in takes roughly $t_0 = 1000$ algorithmic steps.
Next we study the MALA. We choose proposal

$$\Theta^* \sim q(\cdot|\Theta^{(t)}) \overset{(\mathrm{d})}{=} \mathcal{N}\left(\Theta^{(t)} - \frac{\varepsilon^2}{2}\Sigma^{-1}\Theta^{(t)}, \varepsilon^2 \mathbb{1}\right),$$

from which the acceptance probability can easily be calculated, see also (4.17). The parameter $\varepsilon = 0.23$ is fine-tuned so that we obtain an average acceptance rate of roughly 0.574. The trace plot of the first component $(\Theta_1^{(t)})_t$ is given in Figure 4.6 (middle, blue). Note that we choose starting point $\Theta^{(0)} = (10, \ldots, 10)'$ and the burn-in is faster than in the RW-MH algorithm.

Finally, we consider the HMC algorithm. We choose exactly the same $\varepsilon = 0.23$ as in the MALA above, but we merge $L = 5$ steps along the Hamiltonian dynamics. We then apply the following steps. First, we simulate $\widehat{\Psi}^{(t+1)} \sim \mathcal{N}(0, \mathbb{1})$. The leapfrog updates are obtained by initializing $\theta^*(0) = \Theta^{(t)}$ and $\psi^*(0) = \widehat{\Psi}^{(t+1)}$, and updating for $j = 1, \ldots, 5$

$$\begin{aligned} \theta^*(j) &= \left(\mathbb{1} - \frac{\varepsilon^2}{2}\Sigma^{-1}\right)\theta^*(j-1) + \varepsilon\, \psi^*(j-1), \\ \psi^*(j) &= \psi^*(j-1) - \frac{\varepsilon}{2}\,\Sigma^{-1}\left(\theta^*(j-1) + \theta^*(j)\right). \end{aligned}$$

This provides a proposal for the next state $(\Theta^*, \Psi^*) = (\theta^*(5), -\psi^*(5))$. The acceptance probability for this move is then easily calculated from (4.13). For this parametrization we obtain an average acceptance rate of 0.698. The trace plot of the first component $(\Theta_1^{(t)})_t$ is given in Figure 4.6 (rhs, red). The striking difference is that we get much better mixing than in the RH-MH algorithm and in the MALA. This is confirmed by looking at

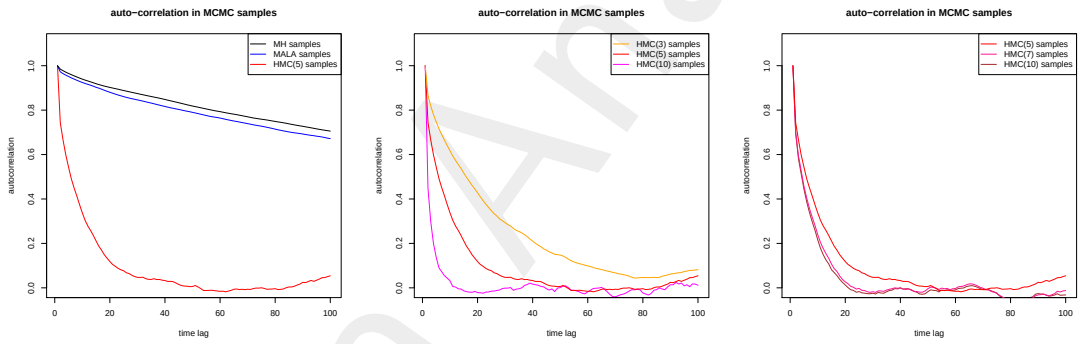


Figure 4.7: Autocorrelations of the first component $\Theta_1^{(t)}$ of $(\Theta^{(t)})_{t=t_0+1,\ldots,10000}$ (after burn-in $t_0 = 1000$) of (lhs) the MH algorithm (black), the MALA (blue) and the HMC algorithm with $L = 5$ (red); (middle) the HMC algorithm for $L = 3, 5, 10$ (orange, red, magenta); and (rhs) the HMC algorithm for constant total move $L\varepsilon = 1.15$ (red, pink, brown).

the resulting empirical auto-correlations in the samples. They are presented in Figure 4.7 (lhs) for the first component of $\Theta^{(t)}$. From this we conclude that we should clearly favor the HMC algorithm with $L > 1$. In Figure 4.8 we also present the resulting QQ-plots (after burn-in $t_0 = 1000$). They confirm the previous statement, in particular, the tails of the HMC algorithm look much better than in the other algorithms.

In the HMC algorithm we still have the freedom of different choices of L and ε. For the above chosen $\varepsilon = 0.23$ we present three different choices of $L = 3, 5, 10$ in Figure 4.9. In this case $L = 10$ has the best mixing properties, also confirmed by the autocorrelation plot in Figure 4.7 (middle). However, L should also not be chosen too large because the

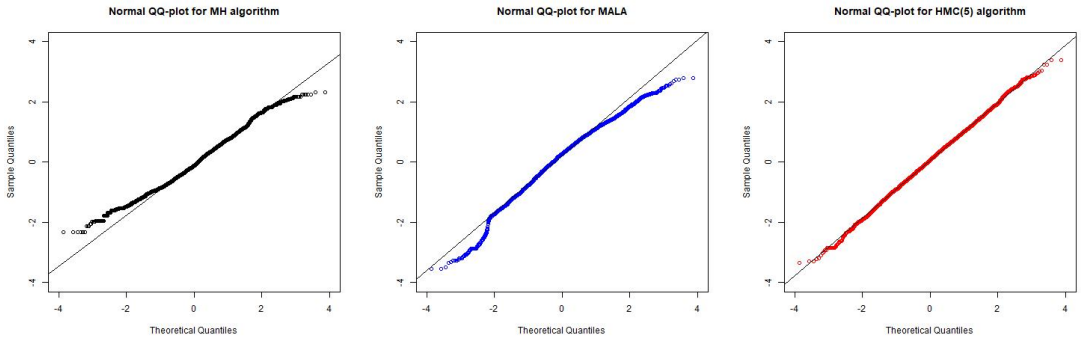


Figure 4.8: QQ-plots of the first component $\Theta_1^{(t)}$ of $(\Theta^{(t)})_{t=t_0+1,\ldots,10000}$ (after burn-in $t_0 = 1000$) of (lhs, black) the MH algorithm, (middle, blue) the MALA, and (rhs, red) the HMC algorithm with $L = 5$.

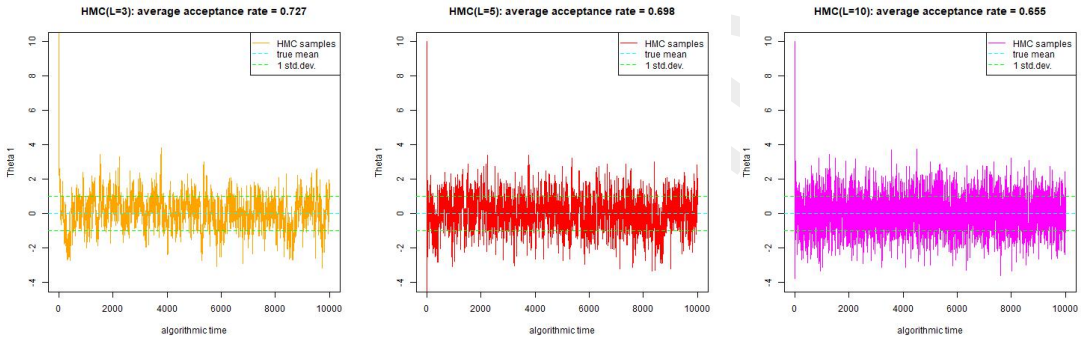


Figure 4.9: Trace plots of the first component $\Theta_1^{(t)}$ of $(\Theta^{(t)})_{t=0,\ldots,10000}$ of the HMC algorithm for (lhs, orange) $L = 3$, (middle, red) $L = 5$, and (rhs, magenta) $L = 10$.

leapfrog update may use too much computational time for large L. Hoffman–Gelman [67] have developed the no-U-turn sampler (NUTS) which fine-tunes L in an adaptive way.

From Figure 4.9 we also observe that the choice of L does not influence the average acceptance probability too much. In Figure 4.10 we chose $L\varepsilon = 5 \cdot 0.23 = 1.15$ constant, i.e. we decrease the step size and increase the number of steps. Of course, this implies that we follow more closely the Hamiltonian dynamics and the average acceptance rate is increasing, see Figure 4.10 from left to right. In Figure 4.7 (rhs) we present the resulting autocorrelation picture. From this we conclude, that $L = 10$ with original step size $\varepsilon = 0.23$, Figure 4.7 (middle, magenta), provides the best mixture of the Markov chain in our example (and should be preferred here). This finishes the example. ■

4.4.6 Markov chain Monte Carlo methods: summary

We have met several simulation algorithms to numerically approximate the density π. Typically, these algorithms are used to determine posterior densities of type (4.4). These posterior densities are known up to the normalizing constants. This then allows one

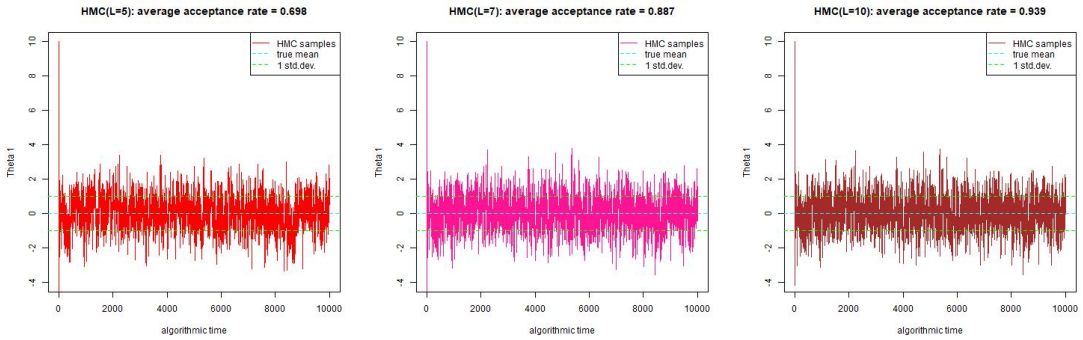


Figure 4.10: Trace plots of the first component $\Theta_1^{(t)}$ of $(\Theta^{(t)})_{t=0,\ldots,10000}$ of the HMC algorithm for (lhs, red) $L = 5$ and $\varepsilon = 0.23$, (middle, pink) $L = 7$ and $\varepsilon = 0.23 \cdot 5/7$, and (rhs, brown) $L = 10$ and $\varepsilon = 0.23/2$, i.e. the total move $L\varepsilon = 1.15$ is constant.

to empirically calculate the posterior Bayesian parameter (4.5). The advantage of the posterior Bayesian parameter is that we receive a natural regularization (determined by the choice of the prior distribution) and, moreover, we can quantify parameter estimation uncertainty through the posterior distribution of this parameter.

Appendix to Chapter 4

4.5 Proofs of Section 4.4

Proof of Lemma 4.12. The MH algorithm provides transition probability for algorithmic time $t \geq 0$

$$p_{t+1|t}(\theta_i, \theta_j) = \mathbb{P}\left[\Theta^{(t+1)} = \theta_j \middle| \Theta^{(t)} = \theta_i\right] = \begin{cases} q(\theta_j|\theta_i)\ \alpha(\theta_i, \theta_j) & \text{for } \theta_i \neq \theta_j, \\ q(\theta_i|\theta_i) + \sum_{\theta_k} q(\theta_k|\theta_i)\,(1 - \alpha(\theta_i, \theta_k)) & \text{for } \theta_i = \theta_j. \end{cases}$$

For the detailed balance condition we need to prove for all θ_i, θ_j and $t \geq 0$

$$\pi(\theta_i)\ p_{t+1|t}(\theta_i, \theta_j) = \pi(\theta_j)\ p_{t+1|t}(\theta_j, \theta_i). \tag{4.15}$$

The case $\theta_i = \theta_j$ is trivial, so consider the case $\theta_i \neq \theta_j$. We assume first that

$$\pi(\theta_j)\ q(\theta_i|\theta_j)\ \geq\ \pi(\theta_i)\ q(\theta_j|\theta_i). \tag{4.16}$$

In this case we have acceptance probability $\alpha(\theta_i, \theta_j) = 1$, and, thus, $p_{t+1|t}(\theta_i, \theta_j) = q(\theta_j|\theta_i)$. We start from the right-hand side in (4.15), note that under (4.16) we have $\alpha(\theta_j, \theta_i) = \pi(\theta_i)\ q(\theta_j|\theta_i)/\pi(\theta_j)\ q(\theta_i|\theta_j) \leq 1$,

$$\begin{aligned} \pi(\theta_j)\ p_{t+1|t}(\theta_j, \theta_i) &= \pi(\theta_j)\ q(\theta_i|\theta_j)\ \alpha(\theta_j, \theta_i) \ =\ \pi(\theta_j)\ q(\theta_i|\theta_j)\ \frac{\pi(\theta_i)\ q(\theta_j|\theta_i)}{\pi(\theta_j)\ q(\theta_i|\theta_j)} \\ &= \pi(\theta_i)\ q(\theta_j|\theta_i) \ =\ \pi(\theta_i)\ p_{t+1|t}(\theta_i, \theta_j). \end{aligned}$$

This proves the first case. For the opposite sign in (4.16) we have $\alpha(\theta_j, \theta_i) = 1$, and we read the last identities from the right to the left as follows

$$\begin{aligned} \pi(\theta_i)\ p_{t+1|t}(\theta_i, \theta_j) &= \pi(\theta_i)\ q(\theta_j|\theta_i)\ \alpha(\theta_i, \theta_j) \ =\ \pi(\theta_i)\ q(\theta_j|\theta_i)\ \frac{\pi(\theta_j)\ q(\theta_i|\theta_j)}{\pi(\theta_i)\ q(\theta_j|\theta_i)} \\ &= \pi(\theta_j)\ q(\theta_i|\theta_j) \ =\ \pi(\theta_j)\ p_{t+1|t}(\theta_j, \theta_i). \end{aligned}$$

This proves the lemma. □

Proof of Lemma 4.13. The acceptance probability in (4.13) for $L = 1$ is

$$\alpha\left((\Theta^{(t)}, \widehat{\Psi}^{(t+1)}), (\Theta^*, \Psi^*)\right) = \min\left(\frac{\pi(\Theta^*)}{\pi(\Theta^{(t)})}\ \exp\left\{-\frac{1}{2}(\Psi^*)' T^{-1} \Psi^* + \frac{1}{2}(\widehat{\Psi}^{(t+1)})' T^{-1} \widehat{\Psi}^{(t+1)}\right\}, 1\right).$$

Note that we have identity

$$z = \Psi^* + \frac{\varepsilon}{2}\ \nabla_\theta \log \pi(\Theta^*) = -\widehat{\Psi}^{(t+1)} - \frac{\varepsilon}{2}\ \nabla_\theta \log \pi(\Theta^{(t)}).$$

We define $a^* = \frac{\varepsilon}{2}\ \nabla_\theta \log \pi(\Theta^*)$ and $a^{(t)} = \frac{\varepsilon}{2}\ \nabla_\theta \log \pi(\Theta^{(t)})$. Then we obtain

$$\begin{aligned} (\Psi^*)' T^{-1} \Psi^* - (\widehat{\Psi}^{(t+1)})' T^{-1} \widehat{\Psi}^{(t+1)} &= (z - a^*)' T^{-1} (z - a^*) - (z + a^{(t)})' T^{-1} (z + a^{(t)}) \\ &= -2(a^* + a^{(t)})' T^{-1} z + (a^*)' T^{-1} a^* - (a^{(t)})' T^{-1} a^{(t)}. \end{aligned}$$

For z we have using the proposal Θ^*

$$z = -\widehat{\Psi}^{(t+1)} - \frac{\varepsilon}{2}\ \nabla_\theta \log \pi(\Theta^{(t)}) = \frac{1}{\varepsilon}\ T\left(\Theta^{(t)} - \Theta^*\right).$$

Therefore, we obtain

$$(\Psi^*)'T^{-1}\Psi^* - (\widehat{\Psi}^{(t+1)})'T^{-1}\widehat{\Psi}^{(t+1)} \;=\; -\frac{2}{\varepsilon}(a^* + a^{(t)})'\left(\Theta^{(t)} - \Theta^*\right) + (a^*)'T^{-1}a^* - (a^{(t)})'T^{-1}a^{(t)}.$$

This provides acceptance probability (4.13) for $L = 1$ being the minimum of 1 and

$$\frac{\pi(\Theta^*)}{\pi(\Theta^{(t)})}\; \exp\left\{\frac{1}{\varepsilon}(a^* + a^{(t)})'\left(\Theta^{(t)} - \Theta^*\right) - \frac{1}{2}(a^*)'T^{-1}a^* + \frac{1}{2}(a^{(t)})'T^{-1}a^{(t)}\right\}.$$

If we run an MH algorithm with proposal (4.14), we obtain acceptance probability being the minimum of 1 and

$$\frac{\pi(\Theta^*)}{\pi(\Theta^{(t)})}\,\frac{\exp\left\{-\frac{1}{2\varepsilon^2}\left(\Theta^{(t)} - \Theta^* - \varepsilon T^{-1}a^*\right)'T\left(\Theta^{(t)} - \Theta^* - \varepsilon T^{-1}a^*\right)\right\}}{\exp\left\{-\frac{1}{2\varepsilon^2}\left(\Theta^* - \Theta^{(t)} - \varepsilon T^{-1}a^{(t)}\right)'T\left(\Theta^* - \Theta^{(t)} - \varepsilon T^{-1}a^{(t)}\right)\right\}}. \tag{4.17}$$

Since the last two displayed formulas are identical, we obtain the same acceptance probability and the claim follows. □

Chapter 5

Neural Networks

5.1 Feed-forward neural networks

Feed-forward neural networks are popular methods in data science and machine learning. Originally, they have been inspired by the functionality of brains and have their roots in the 1940s. In essence, feed-forward neural networks can be understood as high-dimensional non-linear regression functions, and resulting statistical models can be seen as parametric regression models. However, due to their high-dimensionality and non-interpretability of parameters, they are often called non-parametric regression models. Typically, one distinguishes between two types of feed-forward neural networks: (i) *shallow feed-forward neural networks* having one hidden layer, (ii) and *deep feed-forward neural networks* with more than one hidden layer. These will be described in detail in this chapter. This description is based on the q-dimensional feature space $\mathcal{X} \subset \mathbb{R}^q$ introduced in Chapter 2 and we will again analyze regression problems of type (2.1).

Remark. We use neural networks for modeling regression functions. However, neural networks can be seen in a much broader context. They describe a powerful approximation framework to continuous functions. In fact, neural networks provide a general toolbox of basis functions that can be composed to a very flexible approximation framework. This may be seen in a similar way as the splines and MARS frameworks presented in Chapter 3 or as the toolbox of wavelets and other families of basis functions.

5.1.1 Generic feed-forward neural network construction

In this section we describe the generic architecture of feed-forward neural networks. Explicit examples and calibration techniques are provided in the subsequent sections. This section follows Chapter 7 of Wüthrich–Merz [141], and for a more in-depth introduction we refer to this latter reference.

Activation functions

The first important ingredient (hyperparameter) of a neural network architecture is the choice of the *activation function* $\phi : \mathbb{R} \to \mathbb{R}$. Since we would like to approximate non-linear regression functions, these activation functions should be non-linear. Table 5.1

summarizes common choices. The first two examples in Table 5.1 are differentiable (with

sigmoid/logistic activation function	$\phi(x) = (1+e^{-x})^{-1}$	$\phi' = \phi(1-\phi)$
hyperbolic tangent activation function	$\phi(x) = \tanh(x)$	$\phi' = 1-\phi^2$
step function activation	$\phi(x) = \mathbb{1}_{\{x\geq 0\}}$	
rectified linear unit (ReLU) activation function	$\phi(x) = x\mathbb{1}_{\{x\geq 0\}}$	

Table 5.1: Typical choices of (non-linear) activation functions.

easy derivatives). This is an advantage in algorithms for model calibration that involve derivatives.

The hyperbolic tangent activation function satisfies

$$x \;\mapsto\; \tanh(x) \;=\; \frac{e^x - e^{-x}}{e^x + e^{-x}} \;=\; \frac{2e^{2x}}{1+e^{2x}} - 1 \;=\; 2\left(1+e^{-2x}\right)^{-1} - 1. \tag{5.1}$$

The right-hand side uses the sigmoid activation function. The hyperbolic tangent is symmetric w.r.t. the origin, and this symmetry is an advantage in fitting deep network architectures. We illustrate the sigmoid activation function in more detail. It is given by

$$x \;\mapsto\; \phi(x) = \frac{e^x}{1+e^x} = \left(1+e^{-x}\right)^{-1} \;\in\; (0,1). \tag{5.2}$$

Figure 5.1 shows the sigmoid activation function $x \mapsto \phi(wx)$ for $w \in \{1/4, 1, 4\}$ and

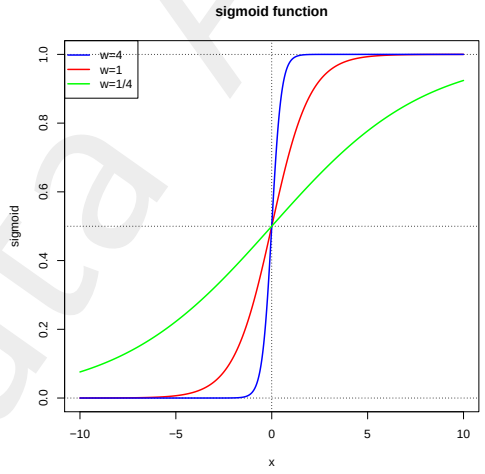


Figure 5.1: Sigmoid activation function $x \mapsto \phi(wx)$ for $w \in \{1/4, 1, 4\}$ and $x \in (-10, 10)$.

$x \in (-10, 10)$. We see that the signal is deactivated for small values of x, and it gets fully activated for big values, and w determines the intensity of activation. The increase of activation takes place in the neighborhood of the origin and it may be shifted by an appropriate choice of translation (also called intercept), i.e. $x \mapsto \phi(w_0 + wx)$ for a given intercept $w_0 \in \mathbb{R}$. The sign of w determines the direction of the activation.

Neural network layers

A neural network consist of (several) neural network layers. Choose hyperparameters $q_{m-1}, q_m \in \mathbb{N}$ and an activation function $\phi : \mathbb{R} \to \mathbb{R}$. A *neural network layer* is a mapping

$$\boldsymbol{z}^{(m)} : \mathbb{R}^{q_{m-1}} \to \mathbb{R}^{q_m} \qquad \boldsymbol{z} \mapsto \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \left(z_1^{(m)}(\boldsymbol{z}), \ldots, z_{q_m}^{(m)}(\boldsymbol{z})\right)',$$

that has the following form. *Neuron* $z_j^{(m)}$, $1 \leq j \leq q_m$, is described by

$$z_j^{(m)} = z_j^{(m)}(\boldsymbol{z}) = \phi\left(w_{j,0}^{(m)} + \sum_{l=1}^{q_{m-1}} w_{j,l}^{(m)} z_l\right) \stackrel{\text{def.}}{=} \phi\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{z}\rangle, \tag{5.3}$$

for given *network weights* $\boldsymbol{w}_j^{(m)} = (w_{j,l}^{(m)})_{0 \leq l \leq q_{m-1}} \in \mathbb{R}^{q_{m-1}+1}$.
Neuron $z_j^{(m)}$ corresponds to a GLM regression w.r.t. feature $\boldsymbol{z} \in \mathbb{R}^{q_{m-1}}$ that consists of a scalar product and then measures the resulting activation of this scalar product in a non-linear fashion ϕ. Function (5.3) is called *ridge function*. A ridge function can be seen as a data compression because it reduces the dimension from q_{m-1} to 1. Since this dimension reduction goes along with a loss of information, we consider simultaneously q_m different ridge functions in a network layer. This network layer has a network parameter $W^{(m)} = (\boldsymbol{w}_1^{(m)}, \ldots, \boldsymbol{w}_{q_m}^{(m)})$ of dimension $q_m(q_{m-1}+1)$.
Often we abbreviate a neural network layer as follows

$$\boldsymbol{z} \;\mapsto\; \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \phi\langle W^{(m)}, \boldsymbol{z}\rangle. \tag{5.4}$$

Feed-forward neural network architecture

A feed-forward neural network architecture is a composition of several neural network layers. The first layer is the *input layer* which is exactly our feature space $\mathcal{X} \subset \mathbb{R}^q$, and we set $q_0 = q$ for the remainder of this chapter. We choose a hyperparameter $d \in \mathbb{N}$ which denotes the *depth* of the neural network architecture. The activations in the *m-th hidden layer* for $1 \leq m \leq d$ are obtained by

$$\boldsymbol{x} \in \mathcal{X} \;\mapsto\; \boldsymbol{z}^{(m:1)}(\boldsymbol{x}) \stackrel{\text{def.}}{=} \left(\boldsymbol{z}^{(m)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \in \mathbb{R}^{q_m}. \tag{5.5}$$

We use the neurons $\boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \in \mathbb{R}^{q_d}$ in the last hidden layer as features in the final regression. We choose the exponential activation function because our responses are unbounded and live on the positive real line. This motivates the *output layer* (regression function)

$$\boldsymbol{x} \in \mathcal{X} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\left\{\beta_0 + \sum_{j=1}^{q_d} \beta_j z_j^{(d:1)}(\boldsymbol{x})\right\} = \exp\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\right\rangle, \tag{5.6}$$

with output network weights $\boldsymbol{\beta} = (\beta_j)_{0 \leq j \leq q_d} \in \mathbb{R}^{q_d+1}$. Thus, (5.6) defines a general regression function which has a *network parameter* $\theta = (\boldsymbol{\beta}, W^{(d)}, \ldots, W^{(1)})$ of dimension $r = q_d + 1 + \sum_{m=1}^{d} q_m(q_{m-1}+1)$. The general aim is to fix a neural network architecture

for the regression model, and fit the resulting network parameter $\theta \in \mathbb{R}^r$ as good as possible to the data.

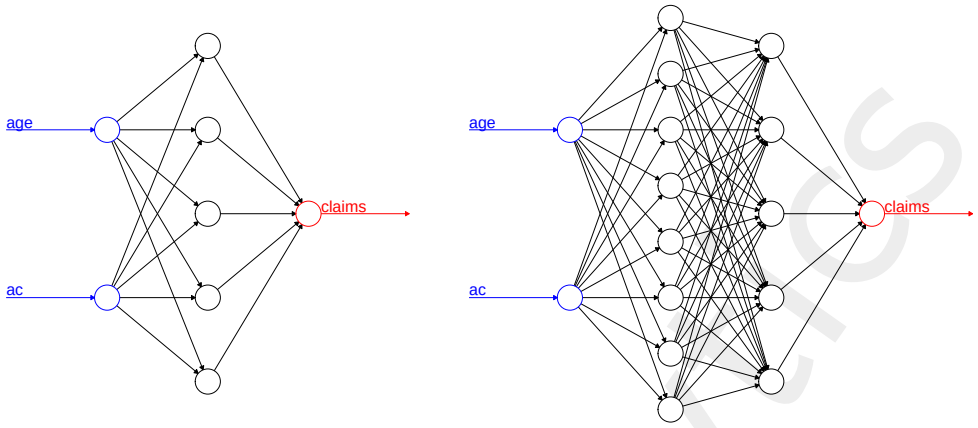


Figure 5.2: (lhs) Neural network of depth $d = 1$ with two-dimensional input feature $\boldsymbol{x} = (\texttt{age}, \texttt{ac})'$ and $q_1 = 5$ hidden neurons; (rhs) neural network of depth $d = 2$ with two-dimensional input feature $\boldsymbol{x} = (\texttt{age}, \texttt{ac})'$ and 2 hidden layers with $q_1 = 8$ and $q_2 = 5$ hidden neurons, respectively; input layer is in blue color, hidden layers are in black color, and output layer is in red color.

If Figure 5.2 we illustrate two different feed-forward neural network architectures. These have network parameters θ of dimensions $r = 21$ and $r = 75$, respectively.

Remarks 5.1.

- The neural network introduced in (5.6) is called *feed-forward* because the signals propagate from one layer to the next (directed acyclic graph). If the network has loops it is called *recurrent* neural network.

- Often, neural networks of depth $d = 1$ are called *shallow networks*, and neural networks of depth $d \geq 2$ are called *deep networks*.

- Here, we have defined a generic feed-forward neural network architecture for a regression problem. Similar architectures can be defined for classification problems. In that case the output layer will have multiple neurons, and often the softmax activation function is chosen to turn the output activations into categorical probabilities for classification.

We briefly summarize the models studied so far which brings a common structure to regression problems. We have studied three different types of regression functions:

Generalized linear models (GLMs): see (2.2),

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle .$$

Generalized additive models (GAMs): see (3.11),

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\langle \boldsymbol{\beta}, \boldsymbol{s}(\boldsymbol{x}) \rangle .$$

Feed-forward neural networks: see (5.6),

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle .$$

In all three examples we choose the log-link function which results in the exponential output activation. In the exponent, we have a scalar product in all three cases. In the first case, the feature $\boldsymbol{x}$ directly runs into the scalar product. As we have seen in Chapter 2 this requires feature engineering so that this log-linear form provides a reasonable regression function for the problem to be solved. In GAMs the linear structure in the exponent is replaced by natural cubic splines which directly pre-process features into an appropriate structure so that we can apply the scalar product on the log-scale. Finally, neural networks allow for even more modeling flexibility in feature engineering $\boldsymbol{x} \mapsto \boldsymbol{z}^{(d:1)}(\boldsymbol{x})$, in particular, w.r.t. interactions as we will see below.

5.1.2 Shallow feed-forward neural networks

We start by analyzing shallow feed-forward neural networks, thus, we choose networks of depth $d = 1$. Figure 5.2 (lhs) shows such a shallow neural network. We start from q_0-dimensional features $\boldsymbol{x} = (x_1, \ldots, x_{q_0})' \in \mathcal{X} \subset \mathbb{R}^{q_0}$ (we have set $q_0 = q$). The hidden layer with q_1 neurons is given by (for notational convenience we drop the upper index $m = 1$ in the shallow network case)

$$\boldsymbol{x} \;\mapsto\; \boldsymbol{z} \;=\; \boldsymbol{z}(\boldsymbol{x}) \;=\; \phi\langle W, \boldsymbol{x} \rangle \;\in\; \mathbb{R}^{q_1}. \tag{5.7}$$

The output layer is then received by

$$\boldsymbol{x} \;\mapsto\; \log \lambda(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{q_1} \beta_j z_j(\boldsymbol{x}) = \langle \boldsymbol{\beta}, \boldsymbol{z}(\boldsymbol{x}) \rangle, \tag{5.8}$$

with output network weights $\boldsymbol{\beta} = (\beta_j)_{0 \le j \le q_1} \in \mathbb{R}^{q_1+1}$. Thus, we have network parameter $\theta = (\boldsymbol{\beta}, W)$ of dimension $r = q_1 + 1 + q_1(q_0 + 1)$.

Universality theorems

Functions of type (5.8) provide a very general family of regression functions (subject to the choice of the output activation). The following statements have been proved by Cybenko [29], Hornik et al. [69] and Leshno et al. [88]: shallow neural networks (under mild conditions on the activation function) can approximate any compactly supported continuous function arbitrarily well (in supremum norm or L^2-norm), if we allow for an

arbitrary number of hidden neurons $q_1 \in \mathbb{N}$. Thus, shallow neural networks are dense in the class of compactly supported continuous functions; and shallow neural networks are sufficiently flexible to approximate any desired (sufficiently regular) regression function. These results are called *universality theorems*. A famous similar statement goes back to Kolmogorov [82] and his student Arnold [3] who were considering Hilbert's 13th problem which is related to the universality theorems. Thus, the universality theorems tell us that shallow neural networks are sufficient, nevertheless we are also going to use deep neural networks, below, because they have a better approximation capacity at lower complexity.

Example 5.2. We provide a simple illustrative example for the functioning of shallow neural networks. We make two different choices for the activation function ϕ. We start by choosing the step function activation, see Table 5.1,

$$\phi(x) = \mathbb{1}_{\{x \geq 0\}}.$$

Note that inside the activation function (5.7) we consider the scalar products $\sum_{l=1}^{q_0} w_{j,l} x_l$ which are translated by intercepts $w_{j,0}$. For the step function activation we are interested in analyzing the property, for $j = 1, \ldots, q_1$,

$$\sum_{l=1}^{q_0} w_{j,l} x_l \overset{?}{\geq} -w_{j,0}.$$

For illustration we choose $q_0 = q_1 = 2$. The blue line in Figure 5.3 (lhs) corresponds

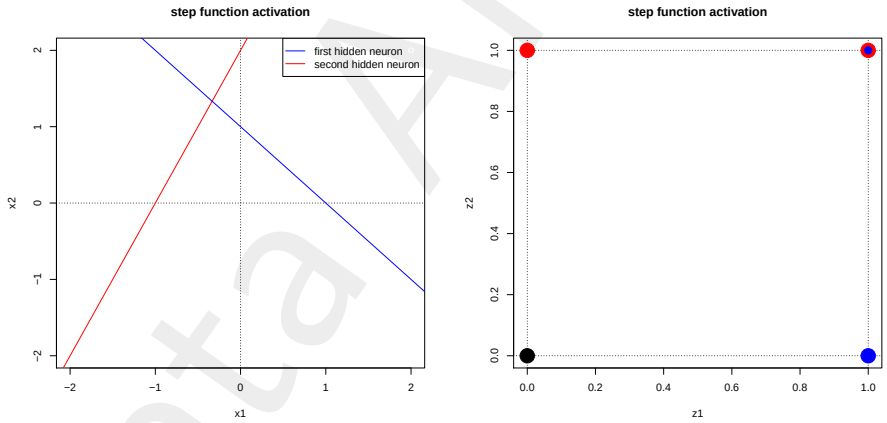


Figure 5.3: Step function activation for $q_0 = q_1 = 2$ (lhs) $\boldsymbol{x} \in \mathbb{R}^2$ and (rhs) $\boldsymbol{z} \in \{0,1\}^2$.

to the translated scalar product of the first hidden neuron $z_1(\boldsymbol{x})$ and the red line to the second hidden neuron $z_2(\boldsymbol{x})$. In our example, all $\boldsymbol{x}$ above the blue line are mapped to $\boldsymbol{z} = (1, z_2)'$ and all $\boldsymbol{x}$ below the blue line to $\boldsymbol{z} = (0, z_2)'$. Similarly, all $\boldsymbol{x}$ above the red line are mapped to $\boldsymbol{z} = (z_1, 1)'$ and all $\boldsymbol{x}$ below the red line to $\boldsymbol{z} = (z_1, 0)'$. Thus, under the step function activation we have four possible values $\boldsymbol{z} \in \{0,1\}^2$ for the neuron activation, see colored dots in Figure 5.3 (rhs). This network can have at most four different expected responses.

As a second example we consider the sigmoid activation function provided in (5.2) for $q_0 = q_1 = 2$, and exactly the same parameters W as in the step function activation of

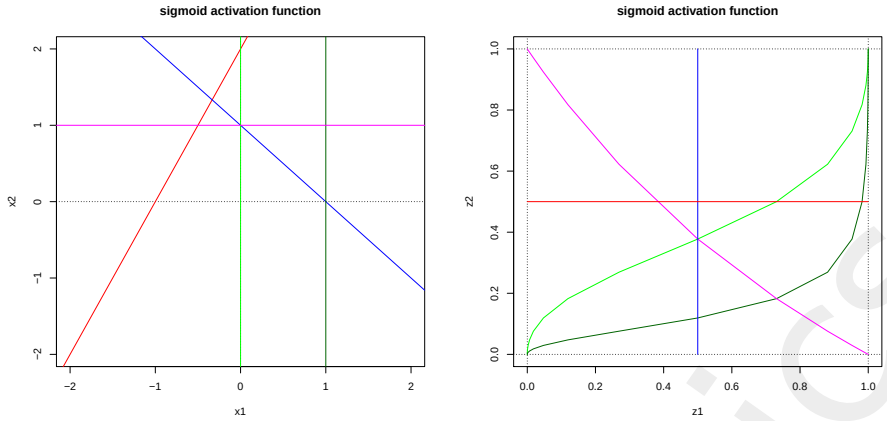


Figure 5.4: Sigmoid activation for $q_0 = q_1 = 2$ (lhs) $\boldsymbol{x} \in \mathbb{R}^2$ and (rhs) $\boldsymbol{z} \in [0,1]^2$.

Figure 5.3. We consider the straight lines in the feature plane $\boldsymbol{x} \in \mathbb{R}^2$ in Figure 5.4 (lhs) and map these lines to the $\boldsymbol{z}$ unit square $[0,1]^2$. The blue line leaves the component $z_1(\boldsymbol{x})$ invariant and the red line leaves the component $z_2(\boldsymbol{x})$ invariant. All other lines that are non-parallel to the blue and the red lines (e.g. the green, dark-green and magenta lines) connect the corners $(0,0)$ and $(1,1)$ or $(1,0)$ and $(0,1)$ of the unit square in the $\boldsymbol{z}$ graph, see Figure 5.4 (rhs).
The ridge function reduces in the scalar product $\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle$ the dimension from q_0 to 1. Observe that $\boldsymbol{x} \mapsto z_j(\boldsymbol{x}) = \phi\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle$ is constant on hyperplanes orthogonal to $\boldsymbol{w}_j$ and the gradient points into the direction of $\boldsymbol{w}_j$, i.e. $\nabla_{\boldsymbol{x}} \phi\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle = \phi'\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle \boldsymbol{w}_j$ (subject to differentiability of the activation function ϕ). ■

Gradient descent method for shallow neural networks

Assume we have chosen the shallow feed-forward neural network (5.7)-(5.8) for our regression modeling problem. The remaining difficulty is to find the optimal network parameter $\theta = (\boldsymbol{\beta}, W)$ of that model. State-of-the-art of neural network calibration uses variants of the *gradient descent method*.

Initial remark. The choices of the activation function ϕ, the depth d of the neural network and the numbers $(q_m)_{1 \le m \le d}$ of hidden neurons are considered to be hyperparameters. Therefore, we fit the network parameter θ conditionally given these hyperparameters. As stated in the universality theorems on page 105, a shallow neural network with an arbitrary number q_1 of neurons is sufficient to approximate any compactly supported continuous function. This illustrates that a large value of q_1 may likely lead to over-fitting of the regression model to the data because we can approximate a fairly large class of functions with that neural network model. This suggests to go for a low dimensional neural network. But this is only half of the story: for model fitting (slight) over-parametrization is often needed in the gradient descent method, otherwise the algorithm may have poor convergence properties. Thus, the crucial point in gradient descent calibration is in most of the cases to stop the optimization algorithm at the right moment

to prevent an over-parametrized model from over-fitting; this is called *early stopping*. We will discuss these issues in more detail below.

We illustrated the gradient descent algorithm on an explicit example. Assume that the expected frequency is modeled by (5.8), and we set

$$\mu(\boldsymbol{x}) = \log \lambda(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{q_1} \beta_j z_j(\boldsymbol{x}) = \langle \boldsymbol{\beta}, \boldsymbol{z} \rangle = \langle \boldsymbol{\beta}, \boldsymbol{z}(\boldsymbol{x}) \rangle,$$

with hidden neurons $\boldsymbol{z}$ satisfying (5.7) for a differentiable activation function ϕ. Under the Poisson assumption

$$N_i \overset{\text{ind.}}{\sim} \text{Poi}\left(\lambda(\boldsymbol{x}_i) v_i\right), \qquad \text{for } i = 1, \ldots, n,$$

we obtain Poisson deviance loss

$$\begin{aligned} D^*(\boldsymbol{N}, \lambda) &= \sum_{i=1}^{n} 2 \left[\lambda(\boldsymbol{x}_i) v_i - N_i - N_i \log\left(\lambda(\boldsymbol{x}_i) v_i\right) + N_i \log N_i\right] \\ &= \sum_{i=1}^{n} 2 \left[e^{\mu(\boldsymbol{x}_i)} v_i - N_i - N_i \mu(\boldsymbol{x}_i) + N_i \log(N_i / v_i)\right]. \end{aligned}$$

The goal is to make this Poisson deviance loss $D^*(\boldsymbol{N}, \lambda)$ small in network parameter θ which is equivalent to determining the MLE for θ. Typically, this optimal network parameter cannot be determined explicitly because the complexity of the problem is too high.[1] The idea therefore is to design an algorithm that iteratively improves the network parameter by looking for locally optimal moves.
We calculate the gradient of the Poisson deviance loss $D^*(\boldsymbol{N}, \lambda)$ w.r.t. θ

$$\nabla_\theta D^*(\boldsymbol{N}, \lambda) = \sum_{i=1}^{n} 2 \left[e^{\mu(\boldsymbol{x}_i)} v_i - N_i\right] \nabla_\theta \mu(\boldsymbol{x}_i) = \sum_{i=1}^{n} 2 \left[\lambda(\boldsymbol{x}_i) v_i - N_i\right] \nabla_\theta \mu(\boldsymbol{x}_i).$$

The last gradient is calculated component-wise. We have

$$\frac{\partial}{\partial \boldsymbol{\beta}} \mu(\boldsymbol{x}) = (z_0(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x})')' = \boldsymbol{z}^+(\boldsymbol{x}) \in \mathbb{R}^{q_1+1}, \tag{5.9}$$

where we set $z_0(\boldsymbol{x}) \equiv 1$ for the intercept β_0. For $j = 1, \ldots, q_1$ and $l = 0, \ldots, q_0$ we have

$$\frac{\partial}{\partial w_{j,l}} \mu(\boldsymbol{x}) = \beta_j \phi'\left(w_{j,0} + \sum_{k=1}^{q_0} w_{j,k} x_k\right) x_l = \beta_j \phi'\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle x_l,$$

where we set $x_0 = 1$ for the intercepts $w_{j,0}$. For the hyperbolic tangent activation function we have $\phi' = 1 - \phi^2$, see Table 5.1. This implies for the hyperbolic tangent activation function

$$\frac{\partial}{\partial w_{j,l}} \mu(\boldsymbol{x}) = \beta_j \left(1 - z_j(\boldsymbol{x})^2\right) x_l. \tag{5.10}$$

Collecting all terms, we obtain the gradient under the hyperbolic activation function choice

$$\nabla_\theta \mu(\boldsymbol{x}) = \left(\boldsymbol{z}^+(\boldsymbol{x})', \beta_1 \left(1 - z_1(\boldsymbol{x})^2\right) x_0, \ldots, \beta_{q_1} \left(1 - z_{q_1}(\boldsymbol{x})^2\right) x_{q_0}\right)' \in \mathbb{R}^r,$$

[1] The issue of not being able to find a global optimizer is discussed in more detail in Remarks 5.3.

with dimension $r = q_1 + 1 + q_1(q_0 + 1)$.
The first order Taylor expansion of the Poisson deviance loss $D^*(\boldsymbol{N}, \lambda)$ in θ is given by, we set $\lambda = \lambda_\theta$ and $\widetilde{\lambda} = \lambda_{\widetilde{\theta}}$,

$$D^*(\boldsymbol{N}, \widetilde{\lambda}) = D^*(\boldsymbol{N}, \lambda) + \nabla_\theta D^*(\boldsymbol{N}, \lambda)' \left(\widetilde{\theta} - \theta\right) + o\left(\|\widetilde{\theta} - \theta\|_2\right),$$

for $\|\widetilde{\theta} - \theta\|_2 \to 0$. Therefore, the negative gradient $-\nabla_\theta D^*(\boldsymbol{N}, \lambda)$ gives the direction for θ of the *maximal local decrease* in Poisson deviance loss.

Assume we are in position $\theta^{(t)}$ after the t-th algorithmic step of the gradient descent method which provides regression function $\boldsymbol{x} \mapsto \lambda^{(t)}(\boldsymbol{x}) = \lambda_{\theta^{(t)}}(\boldsymbol{x})$ with network parameter $\theta^{(t)}$. For a given *learning rate* $\varrho_{t+1} > 0$, the gradient descent algorithm updates this network parameter $\theta^{(t)}$ by

$$\theta^{(t)} \;\mapsto\; \theta^{(t+1)} = \theta^{(t)} - \varrho_{t+1} \nabla_\theta D^*(\boldsymbol{N}, \lambda^{(t)}). \tag{5.11}$$

This update provides new in-sample Poisson deviance loss

$$D^*(\boldsymbol{N}, \lambda^{(t+1)}) = D^*(\boldsymbol{N}, \lambda^{(t)}) - \varrho_{t+1} \left\| \nabla_\theta D^*(\boldsymbol{N}, \lambda^{(t)}) \right\|_2^2 + o\left(\varrho_{t+1}\right),$$

for $\varrho_{t+1} \to 0$. Iteration of this algorithm decreases the Poisson deviance loss step by step.

Remarks 5.3.

- Iteration of algorithm (5.11) does not increase the complexity of the model (which is fixed by the choices of q_0 and q_1); this is different from the gradient boosting machine presented in Chapter 7 on stage-wise adaptive learning, which increases the number of parameters in each iteration of the algorithm.

- By appropriately fine-tuning (tempering) the learning rate $\varrho_{t+1} > 0$, the gradient descent algorithm will converge to a local minimum of the objective function. This may lead to over-fitting for a large hyperparameter q_1 (in an over-parametrized model). Therefore, in neural network calibrations it is important to find a good balance between minimizing in-sample losses and over-fitting. Typically, the data is partitioned into a *training set* and a *validation set*. The gradient descent algorithm is performed on the training set (in-sample), and over-fitting is tracked on the validation set (out-of-sample), and as soon as there are signs of over-fitting on the validation data, the gradient descent algorithm is *early stopped*.

- Early stopped solutions are not unique, even if the stopping criterion is well-defined. For instance, two different initial values (seeds) of the algorithm may lead to different solutions. This issue is a major difficulty in insurance pricing because it implies that "best" prices are not unique in the sense that they may depend on the (random) selection of a seed, for a broader discussion we refer to Section 7.4.4 in [141]. More colloquially speaking this means that we have many competing models of similar quality, and we do not have an objective criterion to select one of them. This is further discussed in Section 5.1.6, below.

- The learning rate $\varrho_{t+1} > 0$ is usually chosen sufficiently small so that the update is still in the region of a locally decreasing deviance loss. Fine-tuning of this learning rate is crucial (and sometimes difficult).

- One may try to use higher order terms in the Taylor expansion (5.11). In most cases this is computationally too costly and one prefers other techniques like momentum-based updates, see (5.12)-(5.13).

- Big data has not been discussed, yet. If the number n of observations in $\mathcal{D}$ is very large, then the gradient calculation $\nabla_\theta D^*$ involves high-dimensional matrix multiplications. This may be very slow in computations. Therefore, typically, the *stochastic gradient descent* (SGD) method is used. The SGD method considers for each step in (5.11) only a (random) sub-sample (*batch*) of all observations in $\mathcal{D}$. In one *epoch* the SGD algorithm runs through all cases in $\mathcal{D}$ once. In applications one usually chooses the *batch size* and the number of epochs the SGD method should be iterated.

  If one considers all observations in $\mathcal{D}$ simultaneously, then one receives the optimal direction and for this reason that latter is sometimes also called *steepest gradient descent* method.

- The initial value for the network parameter in the gradient descent algorithm should be taken at random to avoid any kind of symmetry that may trap the algorithm in a saddle point.

Listing 5.1: Steepest gradient descent method for a shallow neural network

```
1  n      <- nrow(X)              # number of observations in design matrix X
2  z      <- array(1, c(q1+1, n)) # dimension of the hidden layer
3  z[-1,] <- tanh(W %*% t(X))     # initialize neurons for W
4  lambda <- exp(t(beta) %*% z)   # initialize frequency for beta
5  for (t1 in 1:epochs){
6                delta.2   <- 2*(lambda*dat$expo - dat$claims)
7                grad.beta <- z %*% t(delta.2)
8                delta.1   <- (beta[-1,] %*% delta.2) * (1 - z[-1,]^2)
9                grad.W    <- delta.1 %*% as.matrix(X)
10               beta      <- beta - rho[2] * grad.beta/sqrt(sum(grad.beta^2))
11               W         <- W - rho[1] * grad.W/sqrt(sum(grad.W^2))
12               z[-1,] <- tanh(W %*% t(X))
13               lambda <- exp(t(beta) %*% z)
14             }
```

In Listing 5.1 we illustrate the implementation of the gradient descent algorithm for a shallow neural network having hyperbolic tangent activation function. On line 1 we extract the number of observations in the design matrix $\mathfrak{X}$, see also (2.7). On lines 2-4 we initialize the regression function λ for an initial network parameter $\theta^{(0)} = (\boldsymbol{\beta}, W)$ (not further specified in Listing 5.1). Lines 7 and 9 give the gradient of the objective function for the hyperbolic tangent activation function. On lines 10-11 we update the network parameter as in (5.11) with different learning rates for the different layers. Finally, on lines 12-13 we calculate the updated regression function.

Similar to the Newton method one may try to consider second order terms (Hessians) in gradient descent steps. However, this is not feasible computationally. Therefore, one tries to mimic second order terms by momentum-based gradient descent methods, see Rumelhart et al. [117]. Choose a momentum coefficient $\nu \in [0,1)$ and initialize momentum $\mathbf{v}^{(0)} = 0 \in \mathbb{R}^r$.

We replace update (5.11) by

$$\begin{aligned} \mathbf{v}^{(t)} &\mapsto \mathbf{v}^{(t+1)} = \nu \mathbf{v}^{(t)} - \varrho_{t+1} \nabla_\theta D^*(\boldsymbol{N}, \lambda^{(t)}), && (5.12) \\ \theta^{(t)} &\mapsto \theta^{(t+1)} = \theta^{(t)} + \mathbf{v}^{(t+1)}. && (5.13) \end{aligned}$$

For $\nu = 0$ we have the classical gradient descent method, for $\nu > 0$ we also consider previous gradients (with exponentially decaying rates).

Another improvement has been proposed by Nesterov [99]. Nesterov has noticed that (for convex functions) the gradient descent update may behave in a zig-zag manner. He proposed the following adjustment. Initialize $\vartheta^{(0)} = \theta^{(0)}$. Update

$$\begin{aligned} \theta^{(t)} &\mapsto \theta^{(t+1)} = \vartheta^{(t)} - \varrho_{t+1} \nabla_\theta D^*(\boldsymbol{N}, \lambda_{\vartheta^{(t)}}), \\ \vartheta^{(t)} &\mapsto \vartheta^{(t+1)} = \theta^{(t+1)} + \frac{t}{t+3}(\theta^{(t+1)} - \theta^{(t)}). \end{aligned}$$

Note that this is a special momentum update.

The R interface to Keras[2] offers predefined gradient descent methods; for technical details we refer to Sections 8.3 and 8.5 in Goodfellow et al. [56].

Predefined gradient descent methods

- 'adagrad' chooses learning rates that differ in all directions of the gradient and that consider the directional sizes of the gradients ('ada' stands for adapted);
- 'adadelta' is a modified version of 'adagrad' that overcomes some deficiencies of the latter, for instance, the sensitivity to hyperparameters;
- 'rmsprop' is another method to overcome the deficiencies of 'adagrad' ('rmsprop' stands for root mean square propagation);
- 'adam' stands for adaptive moment estimation, similar to 'adagrad' it searches for directionally optimal learning rates based on the momentum induced by past gradients measured by an ℓ^2-norm;[3]
- 'nadam' is a Nesterov [99] accelerated version of 'adam'.

[2] Keras is a user-friendly API to TensorFlow, see `https://tensorflow.rstudio.com/keras/`

[3] We would like to indicate that there is an issue with 'adam'. The explanation of the functioning of 'adam' is based on Lemma 10.3 in Kingma–Ba [81], however this lemma is not correct.

Pre-processing features

We have learned how to pre-process categorical feature components in Section 2.4.1. Choosing a fixed reference level, a categorical feature component with k labels can be transformed into a $(k-1)$-dimensional dummy vector, for an example see (2.16). We will use this dummy coding here for categorical feature components. In the machine learning community often one-hot encoding is preferred because it is (slightly) simpler and the full rank property of the design matrix $\mathfrak{X}$ is not an important property in an over-parametrized model.

For the gradient descent method to work properly (and not get trapped) we also need to pre-process continuous feature components. A necessary property therefore is that all feature components live on a comparable scale, otherwise the gradient is dominated by components that live on a bigger scale. In many cases the so-called *MinMaxScaler* is used. Denote by x_l^- and x_l^+ the minimal and maximal feature value of the continuous feature component x_l. Then we transform this continuous feature components for all cases $1 \leq i \leq n$ by

$$x_{i,l} \;\mapsto\; x_{i,l}^{\text{MM}} = 2\,\frac{x_{i,l} - x_l^-}{x_l^+ - x_l^-} - 1 \;\in\; [-1,1]. \tag{5.14}$$

If the resulting feature values $(x_{i,l}^{\text{MM}})_{1\leq i\leq n}$ cluster in interval $[-1,1]$, for instance, because of outliers in feature values, we should first transform them non-linearly to get more uniform values. For convenience we will drop the upper index in $x_{i,l}^{\text{MM}}$ in the sequel.

Example 5.4 (example SNN1)**.** We consider the same set-up as in Example 2.10 (GLM1) and Example 3.4 (GAM1). That is, we only use the feature components `age` and `ac` in the regression model. We choose two different shallow neural network architectures, a first one having $q_1 = 5$ hidden neurons and a second one having $q_1 = 20$ hidden neurons, respectively. The first one is illustrated in Figure 5.2 (lhs).

In a first analysis we study the convergence behavior of the gradient descent algorithm for these two shallow neural network architectures (having network parameters of dimensions $r = 21$ and $r = 81$, respectively). We therefore partition the data $\mathcal{D}$ into a training set and a validation set of equal sizes. As in Example 3.4 (GAM1) we compress the data because we only have 73 different `age`'s and 36 different `ac`'s which results in at most 2'628 non-empty risk cells. In fact, the training data has 2'120 non-empty risk cells and the validation data has 2'087 non-empty risk cells. We then run the gradient descent algorithm on this compressed data (sufficient statistics).

In order to perform the network calibration we use the R interface to Keras which uses a TensorFlow backend.[4] The corresponding code is provided in Listing 5.2. On line 1 we start the `keras` library and on lines 3-4 we define the design matrix $\mathfrak{X}$ and the offsets $\log(v_i)$. On lines 6-9 we define the shallow neural network, and the output weights $\boldsymbol{\beta}$ are initialized on line 9 to give the homogeneous MLE $\widehat{\lambda}$. On lines 11-13 we merge the network part with the offset, and we choose the exponential response activation function for the output layer (this part does not involve any trainable weights). On lines 15-16 the `model` is defined, the optimizer is specified and the objective function is chosen. Line

[4] see `https://keras.io/backend/`

Listing 5.2: Shallow neural network with q_1 hidden neurons in Keras

```
1  library(keras)
2
3  Design <- layer_input(shape = c(q0), dtype = 'float32', name = 'Design')
4  LogVol <- layer_input(shape = c(1),  dtype = 'float32', name = 'LogVol')
5
6  Network = Design %>%
7       layer_dense(units=q1, activation='tanh',   name='Layer1') %>%
8       layer_dense(units=1,  activation='linear', name='Network',
9            weights=list(array(0, dim=c(q1,1)), array(log(lambda0), dim=c(1))))
10
11 Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
12      layer_dense(units=1, activation=k_exp, name = 'Response', trainable=FALSE,
13                      weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))
14
15 model <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
16 model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')
17
18 summary(model)
19
20 model %>% fit(X.train, Y.train, validation_data=list(X.vali,Y.vali),
21                         epochs=10000, batch_size=nrow(X.train), verbose=0)
```

18 summarizes the model which provides the output of Listing 5.3.

Listing 5.3: Shallow neural network with $q_1 = 5$ in Keras: model summary

```
1  Layer (type)                 Output Shape              Param #   Connected to
2  ==================================================================================
3  Design (InputLayer)          (None, 2)                 0
4  ----------------------------------------------------------------------------------
5  Layer1 (Dense)               (None, 5)                 15        Design[0][0]
6  ----------------------------------------------------------------------------------
7  Network (Dense)              (None, 1)                 6         Layer1[0][0]
8  ----------------------------------------------------------------------------------
9  LogVol (InputLayer)          (None, 1)                 0
10 ----------------------------------------------------------------------------------
11 Add (Add)                    (None, 1)                 0         Network[0][0]
12                                                                   LogVol[0][0]
13 ----------------------------------------------------------------------------------
14 Response (Dense)             (None, 1)                 2         Add[0][0]
15 ==================================================================================
16 Total params: 23
17 Trainable params: 21
18 Non-trainable params: 2
```

On lines 20-21 of Listing 5.2 we fit this model on the training data and we validate it on the validation data. As batch size we choose the size of the observations (steepest gradient descent method) and we run the gradient descent algorithm for 10'000 epochs using the `nadam` optimizer. We illustrate the convergence behavior in Figure 5.5.
We observe that we receive a comparably slow convergence behavior for both shallow neural networks. We do not see any sign of over-fitting after 10'000 gradient descent iterations. Moreover, there is not any visible difference between the two shallow neural networks.
(For no specific reason,) we select the shallow neural network with $q_1 = 20$ hidden neu-

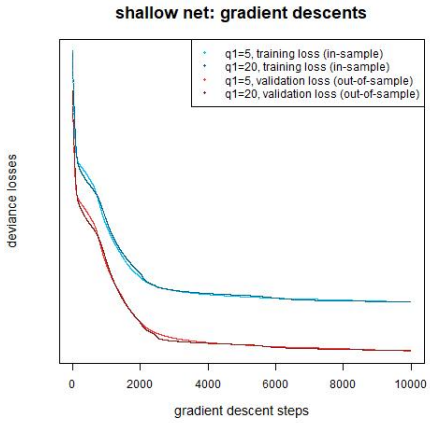


Figure 5.5: Convergence of the gradient descent algorithm for the two shallow neural networks with $q_1 = 5, 20$ hidden neurons; blue gives the training set (in-sample) and red gives the validation set (out-of-sample).

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.1) GLM1	3.1s	11	28.3543	28.3544	0.6052	28.3510	10.2691%
(Ch3.1) GAM1	1.1s	108	28.3248	28.3245	0.5722	28.3134	10.2691%
(Ch5.0) SNN1 $q_1 = 20$	87s	81	–	–	0.5730	28.3242	10.2096%

Table 5.2: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 3.1.

rons, and we fit this network over 20'000 epochs on the entire (compressed) data. We call this model SNN1. The results are illustrated on line (Ch5.0) of Table 5.2. An observation is that the neural network model is better than model GLM1 and comparably good as model GAM1, however, it uses much more computational time for calibration than model GAM1. We interpret the results as follows: (1) Model GLM1 is not competitive because the building of the classes for `age` and `ac` does not seem to have been done in the most optimal way. (2) Model GAM1 is optimal if the feature components `age` and `ac` only interact in a multiplicative fashion, see (3.3). This seems to be the case here,[5] thus, model GAM1 is optimal in some sense, and model SNN1 (only) tries to mimic model GAM1.

In Figure 5.6 we plot the resulting marginal frequencies of the three models GLM1, GAM1 and SNN1. For the feature component `age` we do not see much (visible) differences between models GAM1 and SNN1. For the feature component `ac` the prediction of model

[5] In fact, the discovery that model GAM1 and model SNN1 are similarly good unravels some details of our choice of the true regression function $\lambda^\star$.

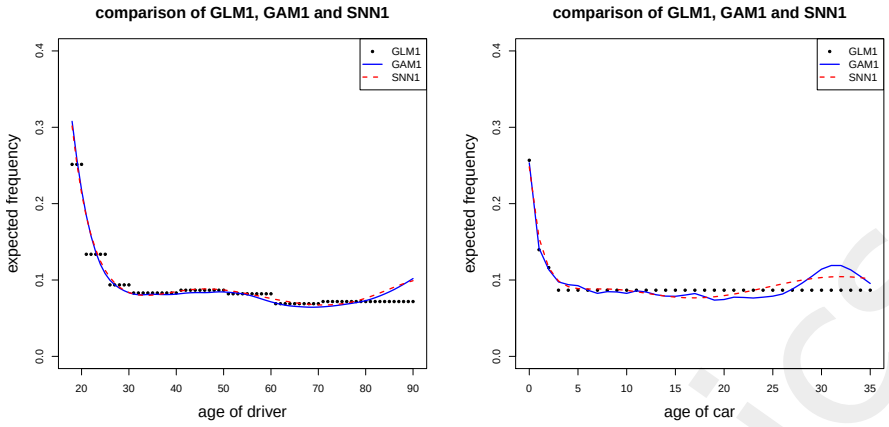


Figure 5.6: Comparison of predicted frequencies in the models GLM1, GAM1 and SNN1.

SNN1 looks like a smooth version of the one of model GAM1. Either model GAM1 overfits for higher ages of cars or model SNN1 is not sufficiently sensitive in this part of the feature space. From the current analysis it is difficult to tell which model is better, and we may work with both of them. Note that these higher ages of cars have very small volumes (years at risk), see Figure A.5, and therefore may be influenced by special configurations of other feature components. This finishes this example. ■

In a next analysis we should incorporate all feature components of the individual cases. Before doing so, we briefly discuss deep feed-forward neural networks. In the previous example we have seen that the gradient descent algorithm may converge very slowly. This may even be more pronounced if we have interactions between feature components because modeling of interactions needs more neurons q_1 in the hidden layer. For this reason, we switch to deep neural networks before discussing the example on all feature components.

5.1.3 Deep feed-forward neural networks

As mentioned above, often deep neural networks are more efficient in model calibration, in particular, if the regression function has interactions between the feature components. For this reason, we study deep feed-forward neural networks in this section.

In view of definition (5.6), feed-forward neural networks of depth $d \geq 2$ are given by

$$\boldsymbol{x} \in \mathcal{X} \;\; \mapsto \;\; \mu(\boldsymbol{x}) = \log \lambda(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{q_d} \beta_j z_j^{(d:1)}(\boldsymbol{x}) = \left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle,$$

with composition $\boldsymbol{z}^{(d:1)}(\boldsymbol{x}) = (\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)})(\boldsymbol{x})$ based on hidden layers $1 \leq m \leq d$

$$\boldsymbol{z} \in \mathbb{R}^{q_{m-1}} \;\; \mapsto \;\; \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \phi\langle W^{(m)}, \boldsymbol{z} \rangle \in \mathbb{R}^{q_m}.$$

Complexity of deep neural networks

Above we have met the universality theorems which say that shallow neural networks are sufficient from a pure approximation point of view. We present a simple example that can easily be reconstructed by a neural network of depth $d = 2$, but it cannot easily be approximated by a shallow one. Choose a two-dimensional feature space $\mathcal{X} = [0,1]^2$ and define the regression function $\lambda : \mathcal{X} \to \mathbb{R}_+$ by

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = 1 + \mathbb{1}_{\{x_2 \geq 1/2\}} + \mathbb{1}_{\{x_1 \geq 1/2,\; x_2 \geq 1/2\}} \;\in\; \{1,2,3\}. \tag{5.15}$$

This regression function is illustrated in Figure 5.7. The aim is to model this regression function with a neural network having step function activation. We rewrite regression

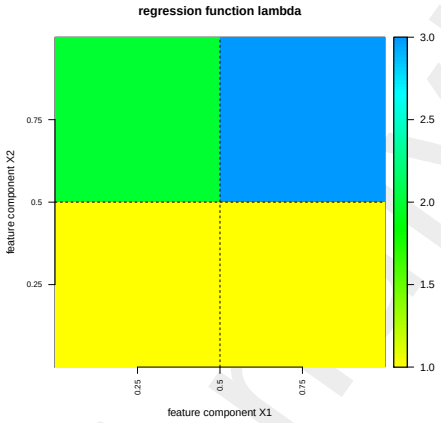


Figure 5.7: Regression function (5.15).

function (5.15) choosing step function activations: for $\boldsymbol{x} \in \mathcal{X}$ we define the first hidden layer

$$\boldsymbol{z}^{(1)}(\boldsymbol{x}) = \left(z_1^{(1)}(\boldsymbol{x}), z_2^{(1)}(\boldsymbol{x})\right)' = \left(\mathbb{1}_{\{x_1 \geq 1/2\}}, \mathbb{1}_{\{x_2 \geq 1/2\}}\right)' \in \{0,1\}^2.$$

This provides us with

$$\begin{aligned}
\lambda(\boldsymbol{x}) &= 1 + \mathbb{1}_{\{x_2 \geq 1/2\}} + \mathbb{1}_{\{x_1 \geq 1/2\}} \mathbb{1}_{\{x_2 \geq 1/2\}} \\
&= 1 + z_2^{(1)}(\boldsymbol{x}) + z_1^{(1)}(\boldsymbol{x})\; z_2^{(1)}(\boldsymbol{x}) \\
&= 1 + \mathbb{1}_{\{z_2^{(1)}(\boldsymbol{x}) \geq 1/2\}} + \mathbb{1}_{\{z_1^{(1)}(\boldsymbol{x}) + z_2^{(1)}(\boldsymbol{x}) \geq 3/2\}} \\
&= 1 + z_1^{(2)}\left(\boldsymbol{z}^{(1)}(\boldsymbol{x})\right) + z_2^{(2)}\left(\boldsymbol{z}^{(1)}(\boldsymbol{x})\right),
\end{aligned}$$

with neurons in the second hidden layer given by

$$\boldsymbol{z}^{(2)}(\boldsymbol{z}) = \left(z_1^{(2)}(\boldsymbol{z}), z_2^{(2)}(\boldsymbol{z})\right)' = \left(\mathbb{1}_{\{z_2 \geq 1/2\}}, \mathbb{1}_{\{z_1 + z_2 \geq 3/2\}}\right)', \qquad \text{for } \boldsymbol{z} \in [0,1]^2.$$

Thus, we obtain deep neural network regression function

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(2:1)}(\boldsymbol{x}) \right\rangle = \left\langle \boldsymbol{\beta}, (\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(\boldsymbol{x}) \right\rangle, \tag{5.16}$$

with output weights $\boldsymbol{\beta} = (1,1,1)' \in \mathbb{R}^3$. We conclude that a deep neural network with $d = 2$ hidden layers and $q_1 = q_2 = 2$ hidden neurons, i.e. with totally 4 hidden neurons,

can perfectly replicate example (5.15). With shallow neural networks there is no similarly efficient way of constructing this regression function with only a few neurons.

Unfortunately, there is no general theory about optimal choices of deep feed-forward neural network architectures. Some examples give optimal numbers of hidden layers under additional assumptions on the feature space, see, for instance, Shaham et al. [122]. An interesting paper is Zaslavsky [144] which proves that q_1 hyperplanes can partition the space $\mathbb{R}^{q_0}$ in at most

$$\sum_{j=0}^{\min\{q_0,q_1\}} \binom{q_1}{j} \qquad \text{disjoint sets.} \tag{5.17}$$

This is an upper complexity bound for shallow neural networks with step function activation. This bound has an exponential growth for $q_1 \leq q_0$, and it slows down to a polynomial growth for $q_1 > q_0$. Thus, with formula (5.17) we can directly bound the complexity of shallow neural networks having step function activation. A lower bound is given in Montúfar et al. [96] for deep neural networks (having ReLU activation function). Shortly said, growing shallow neural networks in width is (much) less efficient in terms of the complexity of the resulting regression functions than growing networks simultaneously in depth and width.

Gradient descent method and back-propagation

Calibration becomes more involved when fitting deep neural networks with many hidden layers, i.e. we cannot simply calculate the gradients as in (5.9)-(5.10). However, there is a nice re-parametrization also known as the back-propagation method. This gives us a recursive algorithm for calculating the gradients. Define for $1 \leq m \leq d+1$ the matrices

$$\mathcal{W}_m = \left(w^{(m)}_{j_m,j_{m-1}}\right)_{j_m=1,\ldots,q_m;j_{m-1}=1,\ldots,q_{m-1}} \in \mathbb{R}^{q_m \times q_{m-1}},$$

where we set $q_{d+1} = 1$ and $w^{(d+1)}_{1,j_d} = \beta_{j_d}$ for $j_d = 0, \ldots, q_d$.

Corollary 5.5 (back-propagation algorithm)**.** *Choose a neural network of depth $d \geq 1$ and with hyperbolic tangent activation function. Denote by $D^*(N,\lambda)$ the Poisson deviance loss of the single case $(N, \boldsymbol{x}, v)$ with expected frequency $\lambda = \lambda(\boldsymbol{x}) = \lambda_\theta(\boldsymbol{x})$ for network parameter $\theta = (\boldsymbol{\beta}, W^{(d)}, \ldots, W^{(1)})$.*

- *Define recursively (back-propagation)*
  - *initialize*
$$\boldsymbol{\delta}^{(d+1)}(\boldsymbol{x}) = 2\left[\lambda(\boldsymbol{x})v - N\right] \in \mathbb{R};$$
  - *iterate for $1 \leq m \leq d$*
$$\boldsymbol{\delta}^{(m)}(\boldsymbol{x}) = \operatorname{diag}\left(1 - (z^{(m:1)}_{j_m}(\boldsymbol{x}))^2\right)_{j_m=1,\ldots,q_m} \mathcal{W}'_{m+1}\, \boldsymbol{\delta}^{(m+1)}(\boldsymbol{x}) \in \mathbb{R}^{q_m}.$$

- *We obtain for* $0 \leq m \leq d$

$$\left(\frac{\partial D^*(N,\lambda)}{\partial w^{(m+1)}_{j_{m+1},j_m}}\right)_{j_{m+1}=1,\ldots,q_{m+1};j_m=0,\ldots,q_m} = \boldsymbol{\delta}^{(m+1)}(\boldsymbol{x})\ (\boldsymbol{z}^{(m:1)}(\boldsymbol{x}))' \in \mathbb{R}^{q_{m+1}\times(q_m+1)},$$

where $\boldsymbol{z}^{(0:1)}(\boldsymbol{x}) = \boldsymbol{x}$ *(including the intercept component).*

For a proof we refer to Proposition 7.4 in [141].

Remarks.

- Corollary 5.5 gives an efficient way of calculating the gradient of the deviance loss function w.r.t. the network parameter. We have provided the specific form of the Poisson deviance loss and the hyperbolic tangent activation function. However, these two items can easily be exchanged by other choices.
- In Listing 5.4 we provide the steepest gradient descent algorithm for a neural network of depth $d = 2$ and using the back-propagation method of Corollary 5.5. This looks similar to Listing 5.1 where we have already been using the back-propagation notation. Our practical applications will again be based on the R interface to Keras library. I.e. we do not need to bother about the explicit implementation, and at the same time we benefit from momentum-based accelerations.

Listing 5.4: Back-propagation for a deep neural network with $d = 2$

```
n       <- nrow(X)                # number of observations in design matrix X
z1      <- array(1, c(q1+1, n))   # dimension of the 1st hidden layer
z2      <- array(1, c(q2+1, n))   # dimension of the 2nd hidden layer
z1[-1,] <- tanh(W1 %*% t(X))      # initialize neurons z1 for W1
z2[-1,] <- tanh(W2 %*% z1)        # initialize neurons z2 for W2
lambda <- exp(t(beta) %*% z2)     # initialize frequency for beta
for (t1 in 1:epochs){
          delta.3   <- 2*(lambda*dat$expo - dat$claims)
          grad.beta <- z2 %*% t(delta.3)
          delta.2   <- (beta[-1,] %*% delta.3) * (1 - z2[-1,]^2)
          grad.W2   <- delta.2 %*% t(z1)
          delta.1   <- (t(W2[,-1]) %*% delta.2) * (1 - z1[-1,]^2)
          grad.W1   <- delta.1 %*% as.matrix(X)
          beta      <- beta - rho[3] * grad.beta/sqrt(sum(grad.beta^2))
          W2        <- W2 - rho[2] * grad.W2/sqrt(sum(grad.W2^2))
          W1        <- W1 - rho[1] * grad.W1/sqrt(sum(grad.W1^2))
          z1[-1,]   <- tanh(W1 %*% t(X))
          z2[-1,]   <- tanh(W2 %*% z1)
          lambda    <- exp(t(beta) %*% z2)
            }
```

Example 5.6 (example DNN1)**.** In this example we consider a claims frequency modeling attempt that is based on all feature components. We therefore implement a deep feed-forward neural network architecture. The results will be comparable to the ones of models GLM4 and GAM3, and they follow up Table 3.2. We use a modeling approach similar to Conclusion 2.8, namely, we consider the 5 continuous feature components `age`, `ac`,

power, area and log(dens), these are transformed with the MinMaxScaler (5.14) to the interval $[-1,1]$; the binary component gas is set to $\pm 1/2$; and the 2 categorical feature components brand and ct are treated by dummy coding. This provides a feature space

$$\mathcal{X} = [-1,1]^5 \times \{-1/2, 1/2\} \times \mathcal{X}^{\texttt{brand}} \times \mathcal{X}^{\texttt{ct}} \subset \mathbb{R}^{q_0}, \tag{5.18}$$

of dimension $q_0 = 5 + 1 + 10 + 25 = 41$.

Listing 5.5: Deep neural network in Keras

```
library(keras)

Design <- layer_input(shape = c(q0), dtype = 'float32', name = 'Design')
LogVol <- layer_input(shape = c(1),  dtype = 'float32', name = 'LogVol')

Network = Design %>%
     layer_dense(units=q1, activation='tanh',   name='Layer1') %>%
     layer_dense(units=q2, activation='tanh',   name='Layer2') %>%
     layer_dense(units=q3, activation='tanh',   name='Layer3') %>%
     layer_dense(units=1,  activation='linear', name='Network',
          weights=list(array(0, dim=c(q3,1)), array(log(lambda0), dim=c(1))))

Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
     layer_dense(units=1, activation=k_exp, name = 'Response', trainable=FALSE,
                      weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))

model <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')

summary(model)
```

Listing 5.6: Deep neural network with $(q_1, q_2, q_3) = (20, 15, 10)$: model summary

```
Layer (type)              Output Shape     Param #  Connected to
==================================================================
Design (InputLayer)       (None, 41)       0
------------------------------------------------------------------
Layer1 (Dense)            (None, 20)       840      Design[0][0]
------------------------------------------------------------------
Layer2 (Dense)            (None, 15)       315      Layer1[0][0]
------------------------------------------------------------------
Layer3 (Dense)            (None, 10)       160      Layer2[0][0]
------------------------------------------------------------------
Network (Dense)           (None, 1)        11       Layer3[0][0]
------------------------------------------------------------------
LogVol (InputLayer)       (None, 1)        0
------------------------------------------------------------------
Add (Add)                 (None, 1)        0        Network[0][0]
                                                    LogVol[0][0]
------------------------------------------------------------------
Response (Dense)          (None, 1)        2        Add[0][0]
==================================================================
Total params: 1,328
Trainable params: 1,326
Non-trainable params: 2
```

We use a feed-forward neural network of depth $d = 3$ having $q_1 = 20$, $q_2 = 15$ and $q_3 = 10$ hidden neurons, respectively. The corresponding R code is given in Listing 5.5;

we mention that this code is almost identical to the one in Listing 5.2 except that it has more hidden layers on lines 7-9. This network is illustrated in Figure 5.9 (lhs), below, and the different colors in the input layer exactly correspond to the ones in (5.18).
In Listing 5.6 we present the summary of this network model having depth $d = 3$ with hidden neurons $(q_1, q_2, q_3) = (20, 15, 10)$. This model has a network parameter θ of dimension $r = 1'326$. The first hidden layer receives 840 parameters; this high number of parameters is strongly influenced by categorical feature components having many different labels. In our case feature component `ct` has 26 different labels which provides $25 \cdot 20 = 500$ parameters for the first hidden layer (dummy coding). In (5.19) on embedding layers we are going to present a different treatment of categorical feature components which (often) uses less parameters (and is related to representation learning).

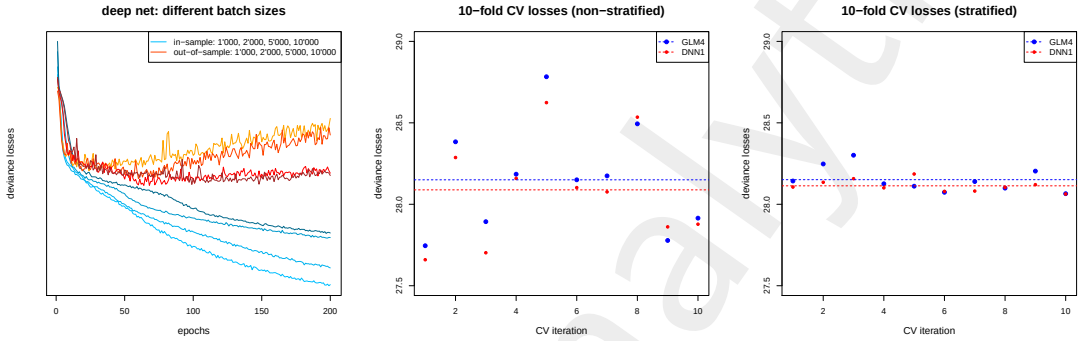


Figure 5.8: (lhs) SGD method on different batch sizes (the more light the color the smaller the batch size) with blue being the training data and red the validation data; (middle, rhs) 10-fold cross-validation losses of models GLM4 and DNN1 on the $K = 10$ individual partitions (non-stratified and stratified) on the identical scale.

We calibrate this deep neural network architecture using the momentum-based gradient descent optimizer `nadam`, see line 18 of Listing 5.5. In the application of this optimizer we still have the freedom of choosing the number of epochs and the batch size (of the stochastic gradient descent (SGD) method). The batch size should be related to the resulting confidence bounds (2.20) which quantify the amount of experience needed to detect structural differences in frequencies. We therefore consider a validation set (out-of-sample analysis) of 10% of the total data $\mathcal{D}$, and we learn the neural network on 90% of the data (training set) using the different batch sizes of 1'000, 2'000, 5'000 and 10'000 policies, and we evaluate the resulting model out-of-sample on the validation data, exactly as described in formula (1.8) on out-of-sample cross-validation. In Figure 5.8 (lhs) we illustrate the resulting decreases in in-sample losses (blue) and out-of-sample losses (red), the more light the color the smaller the batch size.
We observe that for smaller batch sizes of 1'000 to 2'000 cases, the model starts to over-fit after roughly 50 epochs. For bigger batch sizes of 5'000 or 10'000 cases the SGD algorithm seems to over-fit after roughly 100 epochs and it provides overall better results than for smaller batch sizes. For this reason we run the SGD method for 100 epochs on a batch size of 10'000 cases. We call the resulting model DNN1, and we provide the results in Table 5.3, line (Ch5.1). We observe that model DNN1 has a clearly better

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.1) DNN1	56s	1'326	28.1163	28.1216	0.3596	27.9280	10.0348%

Table 5.3: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 3.2.

performance than models GLM4 and GAM3 in terms of the resulting estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of 0.3596 versus 0.4137 and 0.3967, respectively. This illustrates that model DNN1 can capture (non-multiplicative) interactions which have not been considered in models GLM4 and GAM3. This picture is negatively affected by the fact that we receive a (bigger) bias in model DNN1, i.e. the average frequency turns out to be too low. This bias is problematic in insurance because is says that the balance property of Proposition 2.4 is not fulfilled.

From Table 5.3 we see that typically the cross-validation losses between the non-stratified version and the stratified version are very similar (28.1163 vs. 28.1216 for DNN1). In particular, there does not seem to be any value added by considering the stratified version. In Figure 5.8 (middle, rhs) we revise this opinion. Figure 5.8 (middle, rhs) illustrates the 10-fold cross-validation losses on *every* of the $K = 10$ partitions *individually* for the non-stratified version (middle) and the stratified version (rhs). The crucial observation is that the stratified version (rhs) fluctuates much less compared to the non-stratified one (middle). Moreover, from the stratified version we see that there is a systematic improvement from model GLM4 to model DNN1, whereas in the non-stratified version this improvement could also be allocated to (pure) random fluctuations (coming from process uncertainty). This finishes this deep neural network example. ■

In the previous example we have seen that the deep neural network architecture outperforms the GLM and the GAM approaches. This indicates that the latter two models are missing important (non-multiplicative) interactions. In a next step we could explore these interactions to improve the GLM and GAM approaches. Another direction we could pursue is to explore other neural network architectures compared to the one considered in Example 5.6. Our next goal is slightly different, namely, we are going to discuss an other treatment of categorical feature components than the one used in (5.18).

Embedding layers for categorical feature components

In view of Example 5.6 it seems to be inefficient to use dummy coding (or one-hot encoding) for categorical feature component. In fact, this treatment of nominal labels almost seems to be a waste of network parameters. In natural language processing (NLP) one embeds words (or categorical features) into low dimensional Euclidean spaces

$\mathbb{R}^b$. This approach is also called *representation learning* and it has been proposed in the actuarial literature by Richman [108, 109]. We review it in this section.

We use the feature component `brand` as illustration to discuss so-called *embedding layers*. We have 11 different car brands, and one-hot encoding provides the 11 unit vectors $\boldsymbol{x}^{(1h)} \in \mathbb{R}^{11}$ as numerical representation, see Table 2.1. Thus, every car `brand` has its own unit vector and the resulting distances between all different car brands are the same (using the Euclidean norm in $\mathbb{R}^{11}$). The idea of an embedding layer and representation learning is that we embed categorical labels into low dimensional Euclidean spaces and proximity in these spaces should mean similarity for regression modeling. Choose $b \in \mathbb{N}$ and consider an embedding mapping (representation)

$$\boldsymbol{e} : \{\texttt{B1}, \ldots, \texttt{B6}\} \to \mathbb{R}^b, \qquad \texttt{brand} \mapsto \boldsymbol{e}(\texttt{brand}) \stackrel{\text{def.}}{=} \boldsymbol{e}^{\texttt{brand}}. \tag{5.19}$$

If two car brands are very similar, say `B1` and `B3` are very similar (w.r.t. the regression task), then naturally their embedding vectors $\boldsymbol{e}^{\texttt{B1}}$ and $\boldsymbol{e}^{\texttt{B3}}$ should be close together in $\mathbb{R}^b$; we illustrate this in Figure 5.11 (lhs), below, for an embedding dimension of $b = 2$.

Embedding (5.19) can be viewed as an additional (initial) network layer, where each categorical label sends a signal to b embedding neurons $\boldsymbol{e}^{\texttt{brand}} = (e_1^{\texttt{brand}}, \ldots, e_b^{\texttt{brand}})'$, see Figure 5.9 (rhs) for a two-dimensional embedding $b = 2$ of `brand`. The corresponding optimal embedding weights (embedding vectors) will then be learned during model calibration adding an additional layer to the gradient descent and back-propagation method.

In the subsequent analysis we choose separate embedding neurons (representations) for each categorical feature component (`brand` and `ct`), that is, we embed the two categorical feature components into two (parallel) embedding layers $\mathbb{R}^{b_{\texttt{brand}}}$ and $\mathbb{R}^{b_{\texttt{ct}}}$ of dimensions $b_{\texttt{brand}}$ and $b_{\texttt{ct}}$, respectively. These are then concatenated with the remaining feature components, resulting in a feature space after embedding given by

$$\mathcal{X}^{\boldsymbol{e}} = [-1, 1]^5 \times \{-1/2, 1/2\} \times \mathbb{R}^{b_{\texttt{brand}}} \times \mathbb{R}^{b_{\texttt{ct}}} \subset \mathbb{R}^{q_0}, \tag{5.20}$$

of dimension $q_0 = 5 + 1 + b_{\texttt{brand}} + b_{\texttt{ct}}$, see Figure 5.9.

We briefly discuss the resulting dimensions of the network parameter θ for the different modeling approaches. Choose a shallow neural network with $q_1 = 20$ hidden neurons. If we use dummy coding for `brand` and `ct`, the input layer has dimension $q_0 = 41$, see (5.18). This results in a network parameter θ of dimension $r = q_1 + 1 + q_1(q_0 + 1) = 20 + 1 + 20 \cdot (41 + 1) = 861$ for the dummy coding treatment of our categorical variables. Choose embedding dimensions $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$ for the two embedding layers; this results in $11 \cdot 2 + 26 \cdot 2 = 74$ embedding weights. Feature space $\mathcal{X}^{\boldsymbol{e}}$ has dimension $q_0 = 10$. Including the embedding weights this results in a network parameter θ of dimension $r = 20+1+20\cdot(10+1)+74 = 315$, thus, we obtain much less parameters to be calibrated in this embedding layer treatment of our categorical variables. In the remainder of this section we revisit Example 5.6, but we use embedding layers for `brand` and `ct` of different dimensions. This example also shows how embedding layers can be modeled in the R interface to Keras.

Example 5.7 (example DNN2/3 using embedding layers)**.** We revisit Example 5.6 using a feed-forward neural network of depth $d = 3$ having $(q_1, q_2, q_3) = (20, 15, 10)$ hidden

Listing 5.7: Deep neural network with embedding layers in Keras

```
1  Design <- layer_input(shape = c(q00), dtype = 'float32', name = 'Design')
2  Brand  <- layer_input(shape = c(1),   dtype = 'int32',   name = 'Brand')
3  Canton <- layer_input(shape = c(1),   dtype = 'int32',   name = 'Canton')
4  LogVol <- layer_input(shape = c(1),   dtype = 'float32', name = 'LogVol')
5
6  BrandEmb = Brand %>%
7     layer_embedding(input_dim=11, output_dim=1, input_length=1, name='BrandEmb') %>%
8                                 layer_flatten(name='Brand_flat')
9
10 CantonEmb = Canton %>%
11    layer_embedding(input_dim=26, output_dim=1, input_length=1, name='CantonEmb') %>%
12                                layer_flatten(name='Canton_flat')
13
14 Network = list(Design, BrandEmb, CantonEmb) %>% layer_concatenate(name='Conc') %>%
15     layer_dense(units=20, activation='tanh', name='Layer1') %>%
16     layer_dense(units=15, activation='tanh', name='Layer2') %>%
17     layer_dense(units=10, activation='tanh', name='Layer3') %>%
18     layer_dense(units=1, activation='linear', name='Network',
19                 weights=list(array(0, dim=c(10,1)), array(log(lambda0), dim=c(1))))
20
21 Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
22     layer_dense(units=1, activation=k_exp, name = 'Response', trainable=FALSE,
23                 weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))
24
25 model <- keras_model(inputs = c(Design,Brand,Canton,LogVol), outputs = c(Response))
26 model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')
```

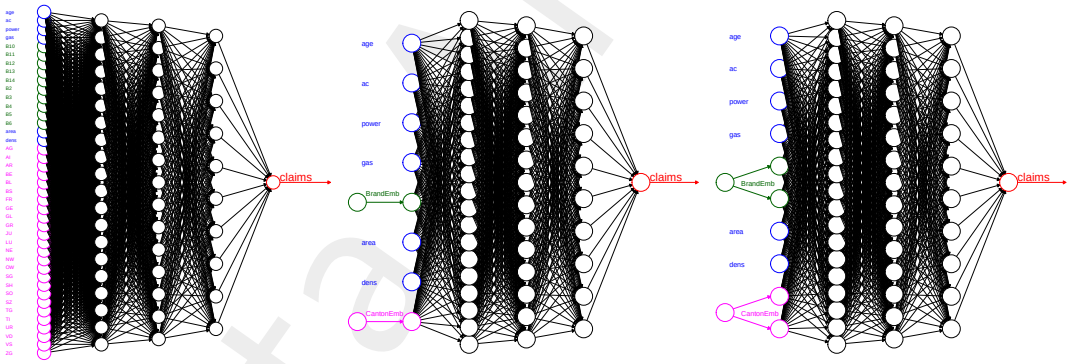


Figure 5.9: Dummy coding with $q_0 = 41$ (lhs), embedding layers for `brand` and `ct` with $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$ (middle) and $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$ (rhs) resulting in $q_0 = 8$ and $q_0 = 10$, respectively.

neurons. The categorical feature components `brand` and `ct` are considered in embedding layers which provides us with feature space $\mathcal{X}^e$ given in (5.20).

We consider two different variants of embedding layers: model DNN2 considers embedding layers of dimension $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$ (Figure 5.9, middle) and model DNN3 of dimension $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$ (Figure 5.9, rhs). In Listing 5.7 we provide the R code for a deep neural network with embedding layers for `brand` and `ct` of dimension $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$, see `output_dim=1` on lines 7 and 11 of Listing 5.7. In Listing 5.8 we present the summary of this model. It has $r = 703$ trainable network parameters which is roughly half as much

Listing 5.8: Deep network with embeddings $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$ and $(q_1, q_2, q_3) = (20, 15, 10)$

```
Layer (type)                 Output Shape       Param #   Connected to
==================================================================================
Brand (InputLayer)           (None, 1)          0
----------------------------------------------------------------------------------
Canton (InputLayer)          (None, 1)          0
----------------------------------------------------------------------------------
BrandEmb (Embedding)         (None, 1, 1)       11        Brand[0][0]
----------------------------------------------------------------------------------
CantonEmb (Embedding)        (None, 1, 1)       26        Canton[0][0]
----------------------------------------------------------------------------------
Design (InputLayer)          (None, 6)          0
----------------------------------------------------------------------------------
Brand_flat (Flatten)         (None, 1)          0         BrandEmb[0][0]
----------------------------------------------------------------------------------
Canton_flat (Flatten)        (None, 1)          0         CantonEmb[0][0]
----------------------------------------------------------------------------------
Conc (Concatenate)           (None, 8)          0         Design[0][0]
                                                          Brand_flat[0][0]
                                                          Canton_flat[0][0]
----------------------------------------------------------------------------------
Layer1 (Dense)               (None, 20)         180       Conc[0][0]
----------------------------------------------------------------------------------
Layer2 (Dense)               (None, 15)         315       Layer1[0][0]
----------------------------------------------------------------------------------
Layer3 (Dense)               (None, 10)         160       Layer2[0][0]
----------------------------------------------------------------------------------
Network (Dense)              (None, 1)          11        Layer3[0][0]
----------------------------------------------------------------------------------
LogVol (InputLayer)          (None, 1)          0
----------------------------------------------------------------------------------
Add (Add)                    (None, 1)          0         Network[0][0]
                                                          LogVol[0][0]
----------------------------------------------------------------------------------
Response (Dense)             (None, 1)          2         Add[0][0]
==================================================================================
Total params: 705
Trainable params: 703
Non-trainable params: 2
```

compared to the dummy coding approach of Listing 5.6.

In a first step, we again determine the optimal batch size for the SGD method. We therefore repeat the out-of-sample analysis on validation data of Example 5.6. We partition the data $\mathcal{D}$ into a training sample of size 90% of the data, and the remaining 10% of the data are used for validation, exactly as described in formula (1.8) on out-of-sample cross-validation. In Figure 5.10 we illustrate the resulting decreases in in-sample losses (blue) and out-of-sample losses (red), the more light the color the smaller the batch size for the two different choices of embedding dimensions $b_{\texttt{brand}} = b_{\texttt{ct}} \in \{1, 2\}$. In general, we observe a slower convergence compared to Figure 5.8 (lhs), and also we prefer a smaller batch size of roughly 5'000 cases.

In Table 5.4 we present the results which are based on 200 epochs with a batch size of 5'000 cases. We observe that the embedding layers substantially increase the quality of the model, compare the estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of 0.1600 and 0.2094 to the corresponding ones of the other models. From this we conclude that we clearly favor deep neural

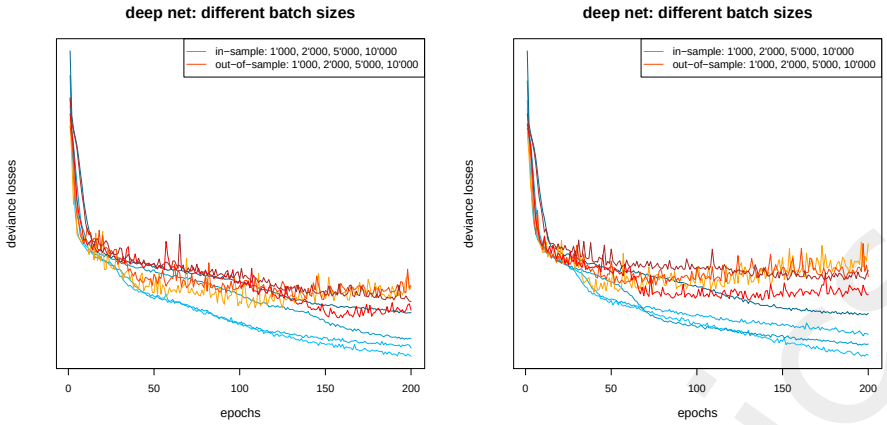


Figure 5.10: SGD method on different batch sizes with embedding layer dimensions (lhs) $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$ and (rhs) $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$; in blue the training losses (in-sample) and in red the validation losses (out-of-sample).

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.1) DNN1	56s	1'326	28.1163	28.1216	0.3596	27.9280	10.0348%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%

Table 5.4: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 5.3.

networks with embedding layers for categorical feature components. However, this again comes at the price of a bigger bias in the average frequency, also compare to Proposition 2.4 on the balance property. Of course, this is problematic in insurance.

K-fold cross-validation is a bit time-consuming in this case. However, the results of Table 5.4 show that we clearly favor the deep neural network with embedding layers, though we cannot decide between the two models DNN2 and DNN3 having different embedding dimensions. Note that optimal embedding dimensions may also differ between the different categorical variables which has not been considered here.

Another nice consequence of low dimensional embedding layers is that they allow us for graphical illustration of the representations learned. In Figure 5.11 we plot the resulting embedding weights $\boldsymbol{e}^{\texttt{brand}} \in \mathbb{R}^2$ and $\boldsymbol{e}^{\texttt{ct}} \in \mathbb{R}^2$ of the categorical variables `brand` and `ct` for embedding dimension $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$. Such a representation may provide clustering among categorical variables, for instance, from Figure 5.11 (lhs) we conclude that car

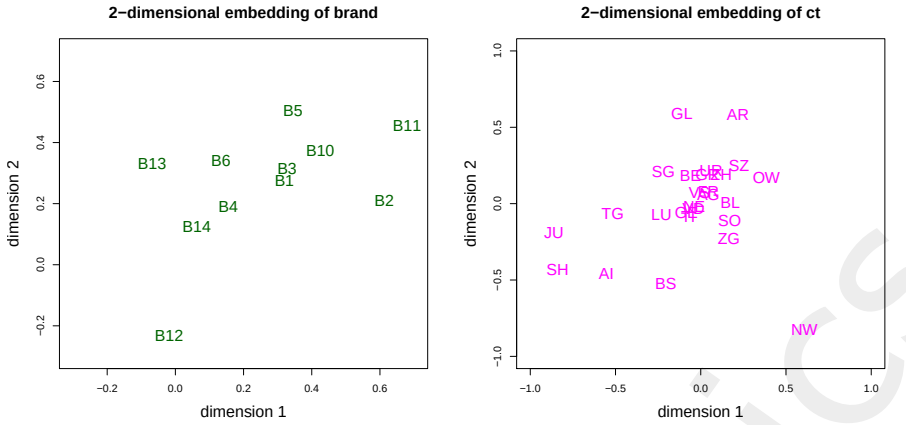


Figure 5.11: Embedding weights $e^{\texttt{brand}} \in \mathbb{R}^2$ and $e^{\texttt{ct}} \in \mathbb{R}^2$ of the categorical variables `brand` and `ct` for embedding dimension $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$.

brand `B12` is very different from all other car brands.

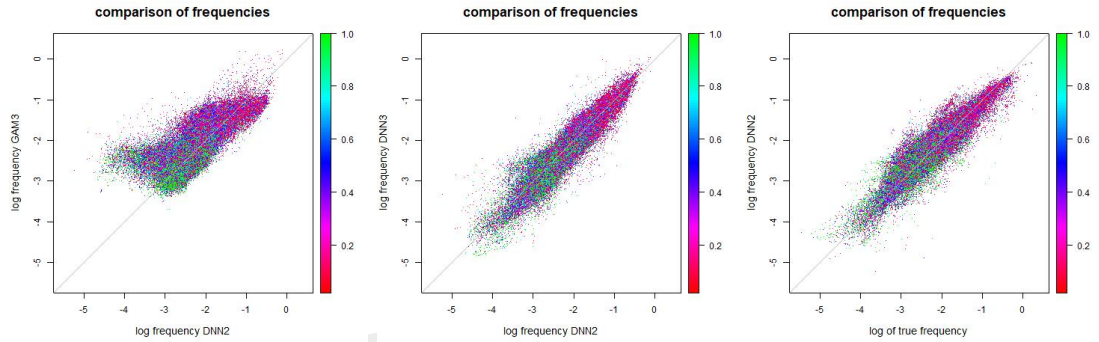


Figure 5.12: Resulting estimated frequencies (on log scale) of models DNN2 vs. GAM3 (lhs), DNN2 vs. DNN3 (middle), and true vs. DNN3 (rhs).

In Figure 5.12 we illustrate the resulting frequency estimates (on log scale) on an individual policy level. The graphs compare models DNN2 vs. GAM3 (lhs), models DNN2 vs. DNN3 (middle), as well as the true log frequency $\log \lambda^\star$ to model DNN2 (rhs). Comparing the graphs on the right-hand side of Figures 3.7 and 5.12, we see a clear improvement in frequency estimation from model GAM3 to model DNN2. Especially, policies with small frequencies are captured much better in the latter model. What might be a bit worrying is that the differences between models DNN2 and DNN3 are comparably large. Indeed, this is a major issue in applications of network predictions that we may receive substantial differences on an individual policy level, which cannot be detected on a portfolio level. These differences may even be induced by choosing different seeds in the SGD calibration method. For a broader discussion we refer to Richman–Wüthrich [111]. This finishes the example. ■

Special purpose layers

Besides the classical hidden feed-forward layers and the embedding layers, there are many other layers in neural network modeling that may serve a certain purpose. We mention some of them.

Drop-out layers

A method to prevent from over-training individual neurons to a certain purpose is to introduce so-called *drop-out layers*. A drop-out layer, say, after 'Layer2' on line 17 of Listing 5.7 would remove during a gradient descent step at random any of the 15 neurons in that layer with a given drop-out probability $\phi \in (0,1)$, and independently from the other neurons. This random removal will imply that the remaining neurons needs to sufficiently well cover the dropped-out neurons. This implies that a single neuron is not over-trained to a certain purpose because it may need to take over several different roles at the same time; we refer to Srivastava et al. [124] and Wager et al. [130].

Normalization layers

We pre-process all feature components of $\boldsymbol{x}$ so that they are living on a comparable scale, see (5.14) for the MinMaxScaler of continuous components. This is important for the gradient descent algorithm to work properly. If we have very deep neural network architectures with, for instance, ReLU activations, it may happen that the activations of the neurons in $\boldsymbol{z}^{(m)}$ again live on different scales. To prevent from this scale-imbalance one may insert *normalization layers* between different hidden layers.

Skip connections

In the next section we are going to meet *skip connections* which are feed-forward connections that skip certain hidden layers. That is, we may decide that a given feature component x_l activates neurons $\boldsymbol{z}^{(1)}$ and that it is at the same time directly linked to, say, neurons $\boldsymbol{z}^{(3)}$, skipping the second hidden layer. Skip connections are often used in very deep feed-forward neural network architectures because they may lead to faster convergence in calibration. In our examples above, we have used a skip connection to model the offset. Note that the volume directly impacts the output layer, skipping all hidden layers, see for instance Listing 5.7, line 21.

Depending on the problem there are many other special layers like convolutional layers and pooling layers in pattern recognition, recurrent layers in time series problems, noise layers, etc.

5.1.4 Combined actuarial neural network approach

The CANN regression model

In this section we revisit the modeling approach proposed in the editorial "Yes, we CANN!" of ASTIN Bulletin [140].[6] The main idea is to **C**ombine a classical **A**ctuarial

[6] In my presentation on "Yes, we CANN!" at the Waterloo Conference in Statistics, Actuarial Science, and Finance on April 25-26, 2019, I have been kindly introduced by Prof. Sheldon Lin (University of

regression model with a **N**eural **N**etwork (CANN) approach. This will allow us to simultaneously benefit from both worlds. The basis of the CANN approach is that the classical actuarial regression model can be brought into a neural network structure. This is the case for many/most parametric regression models.

We illustrate the CANN approach on the GLM example of Chapter 2 and the deep feed-forward neural network of this chapter. Assume that the two models have common feature space $\mathcal{X} \subset \mathbb{R}^{q_0}$. The GLM has regression function, see (2.2),

$$\boldsymbol{x} \;\mapsto\; \lambda^{\mathrm{GLM}}(\boldsymbol{x}) = \exp\left\langle \boldsymbol{\beta}^{\mathrm{GLM}}, \boldsymbol{x} \right\rangle,$$

with GLM parameter $\boldsymbol{\beta}^{\mathrm{GLM}} \in \mathbb{R}^{q_0+1}$, and the deep feed-forward neural network of depth d has regression function, see (5.6),

$$\boldsymbol{x} \;\mapsto\; \lambda^{\mathrm{DNN}}(\boldsymbol{x}) = \exp\left\langle \boldsymbol{\beta}^{\mathrm{DNN}}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle,$$

with network parameter $\theta^{\mathrm{DNN}} = (\boldsymbol{\beta}^{\mathrm{DNN}}, W^{(d)}, \ldots, W^{(1)}) \in \mathbb{R}^{q_d+1+\sum_{m=1}^{d} q_m(q_{m-1}+1)}$.

For the CANN approach we combine these two models, namely, we define the CANN regression function

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\left\{ \left\langle \boldsymbol{\beta}^{\mathrm{GLM}}, \boldsymbol{x} \right\rangle + \left\langle \boldsymbol{\beta}^{\mathrm{DNN}}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle \right\}, \tag{5.21}$$

with joint network parameter $\theta = (\boldsymbol{\beta}^{\mathrm{GLM}}, \theta^{\mathrm{DNN}}) \in \mathbb{R}^r$ having dimension $r = q_0 + 1 + q_d + 1 + \sum_{m=1}^{d} q_m(q_{m-1}+1)$. This is illustrated in Figure 5.13.

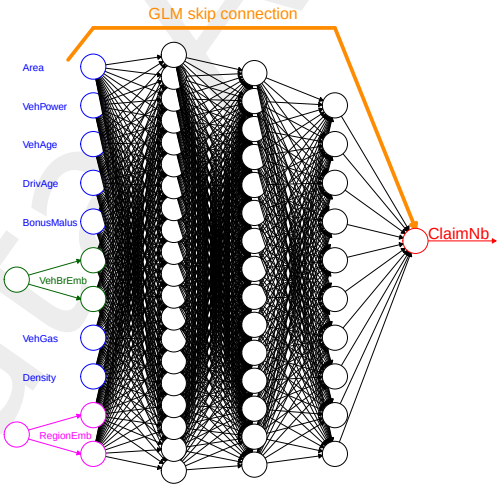


Figure 5.13: Illustration of the CANN approach.

The second ingredient in the CANN approach [140] is a clever fitting strategy: initialize the network parameter $\theta^{(0)} \in \mathbb{R}^r$ for gradient descent so that we exactly start in the classical MLE of the GLM. Denote by $\widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}$ the MLE of $\boldsymbol{\beta}^{\mathrm{GLM}}$ given in Proposition 2.2.

Toronto). In his introduction Prof. Lin suggested that I may also rename my title to "Let's make Actuarial Science great again!".

Start the gradient descent algorithm (5.11) for network calibration of the CANN regression model (5.21) in

$$\theta^{(0)} = \left(\widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}, \boldsymbol{\beta}^{\mathrm{DNN}} \equiv 0, W^{(d)}, \ldots, W^{(1)}\right) \in \mathbb{R}^r. \tag{5.22}$$

Remarks 5.8.

- For initialization (5.22), the gradient descent algorithm then tries to improve the GLM using network features. If the resulting loss substantially decreases during the gradient descent algorithm, then the GLM can be improved, otherwise the GLM is already good. In Chapter 7 we come back to this idea called *boosting*, which tries to improve models in a stage-wise adaptive way.
- The MLE $\widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}$ can either be declared to be trainable or non-trainable. In the latter case we build the network around the GLM compensating for weaknesses of the GLM.

The CANN regression approach in the Poisson case

The CANN regression model (5.21) can easily be implemented similar to Listing 5.7. Though, in some cases this might be a bit cumbersome. The Poisson regression model even admits for a much simpler implementation if we keep the (initial) value $\widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}$ as non-trainable for the GLM parameter $\boldsymbol{\beta}^{\mathrm{GLM}}$ in (5.21).

In that case we have for all cases $(N_i, \boldsymbol{x}_i, v_i)$, $i = 1, \ldots, n$,

$$\begin{aligned} N_i &\overset{\text{ind.}}{\sim} \mathrm{Poi}\left(\exp\left\{\left\langle \widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}, \boldsymbol{x}_i \right\rangle + \left\langle \boldsymbol{\beta}^{\mathrm{DNN}}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle\right\} v_i\right) \\ &\overset{(\mathrm{d})}{=} \mathrm{Poi}\left(\exp\left\langle \boldsymbol{\beta}^{\mathrm{DNN}}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle \widehat{v}_i\right) \overset{(\mathrm{d})}{=} \mathrm{Poi}\left(\lambda^{\mathrm{DNN}}(\boldsymbol{x}_i)\widehat{v}_i\right), \end{aligned} \tag{5.23}$$

where we have defined the *working weights*

$$\widehat{v}_i = \exp\left\langle \widehat{\boldsymbol{\beta}}^{\mathrm{GLM}}, \boldsymbol{x}_i \right\rangle v_i. \tag{5.24}$$

Thus, in this case the CANN regression model calibration is identical to the classical feed-forward neural network calibration, only replacing the original volumes (years at risk) v_i by the working weights $\widehat{v}_i$. In more statistical terms $\log(\widehat{v}_i)$ acts as an *offset* in the regression model.

Remarks 5.9.

- Observe that in (5.24) we use the GLM estimate for the modification of the years at risk v_i. Of course, we can replace the GLM by any other regression model, for instance, we may use the GAM instead. Thus, (5.23)-(5.24) provides a very general way of back-testing (boosting) any regression model adding neural network features to the original regression model. Boosting ideas are going to be discussed in detail in Chapter 7.

- In approach (5.23)-(5.24), the GLM is fully non-trainable. This could be modified by a credibility weight where we declare the credibility weight to be trainable. In practical implementations this means that the offset on lines 21-23 of Listing 5.7 may be declared to be (partially) trainable, too.

Example 5.10 (example CANN1/2)**.** In this example we boost model GAM3 of Example 3.5 with a neural network (in a CANN approach). We have seen that model GAM3 is optimal if we only allow for multiplicative interactions. We aim at exploring other forms of interactions in this example. We use the frequency estimates $\widehat{\lambda}^{\text{GAM3}}(\boldsymbol{x}_i)$ of model GAM3 to receive the working weights $\widehat{v}_i = \widehat{\lambda}^{\text{GAM3}}(\boldsymbol{x}_i)v_i$, similarly to (5.24), and we declare this part to be non-trainable (offset). We then calibrate a deep feed-forward neural network as in (5.23). We choose the network of Example 5.7 having depth $d = 3$ with $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons. The categorical feature components `brand` and `ct` are considered in embedding layers which provides feature space (5.20). This CANN approach is illustrated in Figure 5.13, if we replace the GLM skip connection (in orange color) by a GAM skip connection. This regression model can be calibrated with the R code given in Listing 5.7 replacing the years at risk v_i by the working weights $\widehat{v}_i$. For model calibration we run the SGD method over 50 epochs on batch sizes of 10'000 cases for embedding layer dimensions $b_{\texttt{brand}} = b_{\texttt{ct}} = 1$ (called model CANN1) and $b_{\texttt{brand}} = b_{\texttt{ct}} = 2$ (called model CANN2). Note that after roughly 50 epochs the model starts to over-fit to the training data. This is much faster than in Example 5.7 where we have been running the gradient descent algorithm for 200 epochs. The reason for this smaller number of necessary epochs is that the initial value $\theta^{(0)}$ received from model GAM3 is already reasonable, therefore less gradient descent steps are needed. On the other hand, we also see that the GAM3 calibration is rather good because it takes the gradient descent algorithm roughly 10 epochs to leave the GAM3 solution, i.e. in the first 10 SGD epochs the objective function only decreases very little.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.4) CANN1	28s	703†	27.9292	27.9362	0.2284	27.8940	10.1577%
(Ch5.5) CANN2	27s	780†	27.9306	27.9456	0.2092	27.8684	10.2283%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%

Table 5.5: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters († only considers the network parameters and not the non-trainable GAM parameters), this table follows up from Table 5.4.

The results are presented in Table 5.5. We observe that the CANN approach leads to a substantial improvement of model GAM3, the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ is roughly halved. This indicates that we are missing important (non-multiplicative) interactions

in the GAM regression function. Moreover, the results of the CANN and the DNN approaches are comparably good, however, the CANN models need much less run time for calibration. This finishes this example. ■

In the previous example we have boosted model GAM3 by neural network features. We could also think of a more modular approach for detecting explicit weaknesses in a chosen model. This is exactly what we are going to describe in the next example.

Example 5.11 (exploring missing interactions)**.** In this example we use the framework (5.23)-(5.24) to explore which pairwise (non-multiplicative) interactions are missing in model GAM3. Denote the estimated regression function of model GAM3 by $\boldsymbol{x} \mapsto \widehat{\lambda}^{\mathrm{GAM3}}(\boldsymbol{x})$. The working weights are then given by $\widehat{v}_i = \widehat{\lambda}^{\mathrm{GAM3}}(\boldsymbol{x}_i)v_i$. For testing missing pairwise interactions between, say, components x_l and x_k of $\boldsymbol{x}$, we modify the CANN regression function $\boldsymbol{x} \mapsto \lambda^{\mathrm{DNN}}(\boldsymbol{x})$ in (5.23) as follows

$$(x_l, x_k) \;\; \mapsto \;\; \lambda^{\mathrm{II}}(x_l, x_k) = \exp\left\langle \boldsymbol{\beta}^{\mathrm{II}}, \boldsymbol{z}^{(d:1)}(x_l, x_k) \right\rangle. \tag{5.25}$$

Thus, we restrict the input space to the two variables x_l and x_k under consideration. For this neural network module we use the same architecture of depth $d = 3$ as in the previous examples with embedding layers of dimension 2 for the categorical feature components, and we run the SGD algorithm over 50 epochs on batch sizes of 10'000 cases.

	age	ac	power	gas	brand	area	dens	ct
age		28.1055	28.1055	28.1055	28.0783	28.1055	28.1055	28.1056
ac	0.3966		28.0796	28.0976	28.0071	28.1039	28.1028	28.1055
power	0.3968	0.3718		28.1029	28.1055	28.1130	28.1044	28.1057
gas	0.3966	0.3935	0.3965		28.1059	28.1056	28.1059	28.1055
brand	0.3904	0.3203	0.3968	0.3976		28.1058	28.1055	28.1057
area	0.3966	0.3950	0.4091	0.3965	0.3977		28.1053	28.1055
dens	0.3966	0.3939	0.3962	0.3993	0.3969	0.3964		28.1055
ct	0.3971	0.3967	0.3964	0.3965	0.3973	0.3966	0.3965	

Table 5.6: In-sample Poisson deviance losses (top-right) and estimation losses (bottom-left) of pairwise interaction improvements of model GAM3.

The results are presented in Table 5.6. The upper-right part shows the resulting in-sample losses after considering pairwise interactions, the base value of model GAM3 is 28.1055, see line (Ch3.3) of Table 5.5. The lower-left part shows the resulting estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ after considering pairwise interactions, the base value of model GAM3 is 0.3967. The numbers in red color show pairwise interactions that lead to a substantial decrease in the corresponding loss functions. This indicates that model GAM3 should be enhanced by such non-multiplicative interactions, in particular, an additional interaction between the feature components `ac` and `brand` can substantially improve the regression model. The advantage of this neural network boosting (5.25) is that we do not need to assume any functional form for the missing interactions, but the neural network module is capable to find an appropriate functional form (if necessary).
We could now explore other interactions, for instance, between three feature components, etc. We refrain from exploring more missing properties, but we refer to Section 3.5 of Schelldorfer–Wüthrich [119] for another illustrative example. This finish our example. ■

5.1.5 The balance property in neural networks

We briefly recall the balance property studied in Proposition 2.4. In the GLM framework of Chapter 2 we consider a design matrix $\mathfrak{X} \in \mathbb{R}^{n\times(q+1)}$ of full rank $q+1 \leq n$. The first column of this design matrix is identically equal to 1 modeling the intercept β_0 of the GLM regression function

$$\boldsymbol{x} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle,$$

with regression parameter $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$, see (2.2) and (2.7). Using the Poisson deviance loss $D^*(\boldsymbol{N}, \lambda)$ as objective function we receive a convex optimization problem in a minimal representation, which implies that we have a unique MLE $\widehat{\boldsymbol{\beta}}$ for $\boldsymbol{\beta}$ in this GLM context. For the intercept component this results in requirement

$$\frac{\partial}{\partial \beta_0} D^*(\boldsymbol{N}, \lambda) = \sum_{i=1}^n 2\left[v_i \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i\rangle - N_i\right] \;\stackrel{!}{=}\; 0. \tag{5.26}$$

Thus, a critical point of the intercept component β_0 implies that the balance property is fulfilled. Remark that for exactly this reason one should exclude the intercept from any regularization, see Remarks 4.10. In neural network calibration we do not pay any attention to this point, in fact, we apply early stopping to the gradient descent algorithm. As a consequence we do not stop the algorithm in a critical point of the intercept component and an identity similar to (5.26) typically fails to hold, see for instance the last column of Table 5.5.

Of course, it is not difficult to correct for this deficiency. We are going to discuss this along the lines of Wüthrich [139]. We come back to the deep feed-forward neural network (5.6) given by

$$\boldsymbol{x} \in \mathcal{X} \;\mapsto\; \lambda(\boldsymbol{x}) = \exp\left\{\beta_0 + \sum_{j=1}^{q_d} \beta_j z_j^{(d:1)}(\boldsymbol{x})\right\} = \exp\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\right\rangle.$$

By considering this expression we see that we have a classical GLM in the output layer based on modified features (pre-processed features)

$$\boldsymbol{x} \in \mathcal{X} \;\mapsto\; \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) = \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \in \mathbb{R}^{q_m}.$$

Thus, the neural network $\boldsymbol{x} \mapsto \boldsymbol{z}^{(d:1)}(\boldsymbol{x})$ does a feature engineering which is then used in a GLM step.

Assume that $\widehat{\theta}^{\mathrm{GDM}} = (\widehat{\boldsymbol{\beta}}^{\mathrm{GDM}}, \widehat{W}^{(d)}, \ldots, \widehat{W}^{(1)})$ is an early stopped gradient descent calibration which provides neural network pre-processing $\boldsymbol{x} \mapsto \widehat{\boldsymbol{z}}^{(d:1)}(\boldsymbol{x})$. This motivates to define the pre-processed design matrix

$$\widehat{\mathfrak{X}} = \left(\widehat{z}_l^{(d:1)}(\boldsymbol{x}_i)\right)_{1\leq i\leq n; 0\leq l\leq q_d} \in \mathbb{R}^{n\times(q_d+1)},$$

where we set (again) $\widehat{z}_0^{(d:1)}(\boldsymbol{x}_i) = 1$, for all $1 \leq i \leq n$, modeling the intercept component β_0 of $\boldsymbol{\beta} \in \mathbb{R}^{q_d+1}$. Based on this design matrix $\widehat{\mathfrak{X}}$ we can run a classical GLM receiving a unique MLE $\widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$ from the following requirement, see also Proposition 2.2,

$$\widehat{\mathfrak{X}}' V \exp\{\widehat{\mathfrak{X}}\boldsymbol{\beta}\} \;\stackrel{!}{=}\; \widehat{\mathfrak{X}}' \boldsymbol{N}, \tag{5.27}$$

whenever $\widehat{\mathfrak{X}}$ has full rank $q_d + 1 \leq n$. In neural network terminology this means that for the final calibration step we freeze all network weights $(\widehat{W}^{(d)}, \ldots, \widehat{W}^{(1)})$ and we only optimize over $\boldsymbol{\beta}$ which is a classical convex/concave optimization problem (depending on the sign), and it can be solved by Fisher's scoring method or the IRLS algorithm. This provides improved network parameter $\widehat{\theta}^{\text{GDM+}} = (\widehat{\boldsymbol{\beta}}^{\text{GLM}}, \widehat{W}^{(d)}, \ldots, \widehat{W}^{(1)})$, where the improvement has to be understood in-sample because it could lead to over-fitting. Over-fitting can be controlled by either a more early stopping rule and/or a low dimensional last hidden layer $\boldsymbol{z}^{(d)}$.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.1) DNN1	56s	1'326	28.1163	28.1216	0.3596	27.9280	10.0348%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%
(Ch5.6) DNN3 (balance)	135s	780	–	–	0.2006	27.7509	10.2691%

Table 5.7: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 5.4.

Example 5.12 (bias regularization in neural networks)**.** We revisit model DNN3 provided in Table 5.4. This model considers a deep neural network of depth $d = 3$ with hidden neurons $(q_1, q_2, q_3) = (20, 15, 10)$. For the two categorical features `brand` and `ct` it uses 2-dimensional embedding layers, see Figure 5.9 (rhs). We take the DNN3 calibration provided on line (Ch5.3) of Table 5.4 and we apply the additional GLM step (5.27) for bias regularization (the balance property). Observe that the model DNN3 provides an average frequency of 9.6908% which is (much) smaller than the balance property 10.2691%.

The results are presented on line (Ch5.6) of Table 5.7. We observe the mentioned in-sample improvement, the balance property, and in this case also an estimation loss improvement which says that in the present situation the additional GLM step (5.27) provides a real model improvement. In fact, these results are rather promising and they suggest that we should always explore this additional GLM step. ■

5.1.6 Network ensemble

In the previous example we have seen that the issue of the failure of the balance property in neural network calibrations can easily be solved by an additional GLM step (5.27). In the next example we further explore this issue. The key idea is to build several neural network calibrations for the same architecture. Averaging over these calibrations should

provide better predictive models. This intuition is inspired by the ensemble learning methods presented in Section 7.2, below.

Example 5.13 (network blending/nagging predictor)**.** In Example 5.12 we have provided a solution to be compliant with the balance property in neural network calibrations. The purpose of this example is two fold. First we would like to study how severe the failure of the balance property may be. Therefore, we choose the neural network architecture of model DNN3 and we fit this model to the data $M = 50$ times with 50 different seeds. Early stopping will provide 50 different neural network calibrations for the same architecture which are comparably good in terms of in-sample losses. The first purpose of this example is to analyze the volatility received in the balance property. The second purpose of this example is to see what happens if we blend these models by averaging over the resulting predictors resulting in the nagging predictor.
We run the gradient descent algorithm $M = 50$ times on the same network architecture. We choose all hyperparameters identical, this includes the network architecture, the gradient descent algorithm, its parameters (number of `epochs` $= 200$, `batch size` $= 5'000$, `nadam` optimizer, etc.) and the split in training and validation data. The only difference between the 50 different runs is the choice of the initial seed $\theta^{(0)}$ of the network parameter, see also (5.11).

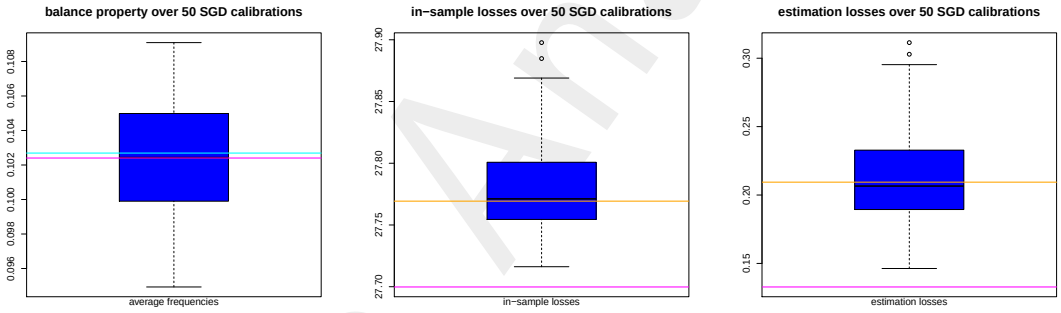


Figure 5.14: Gradient descent calibrations of the same network architecture using $M = 50$ different seeds: (lhs) average frequencies, (middle) in-sample losses on training data, (rhs) estimation losses; orange line corresponds to model DNN3 of Table 5.7, magenta line corresponds to the blended model.

In Figure 5.14 (lhs) we show the box plot of the resulting average frequencies over these 50 calibrations. We observe considerable differences between the different estimated models. The average frequency over all 50 runs is 10.2404% (magenta line), thus, slightly smaller than the balance property of 10.2691% (cyan line). However, the standard deviation in these observations is 0.3539% which is unacceptably high. If we allow for this uncertainty then we can easily charge a premium (on portfolio level) that is 5% too high or too low! Thus, we may receive very accurate prices on a policy level but we may misspecify the price on the portfolio level if we do not pay attention to the balance property.
Figure 5.14 (middle, rhs) show the box plots of the in-sample losses $\mathcal{L}_D^{\text{is}}$ on the training data and the estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$, respectively, over the 50 different seeds; the

orange lines show the figures of model DNN3 in Table 5.7. From these plots we conclude that model DNN3 has a rather typical behavior because the loss figures over the 50 calibrations spread around the ones of model DNN3.
Now comes the interesting fun part. The $M = 50$ different runs have provided 50 network calibrations $\widehat{\lambda}^{(m)}(\cdot)$, $m = 1, \ldots, M$, always on the same network architecture. A natural idea is to consider the ensemble of these models by averaging them. This motivates the (blended) regression function called *nagging predictor* in Richman–Wüthrich [111]

$$\boldsymbol{x} \;\mapsto\; \widehat{\lambda}^{\text{blend}}(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} \widehat{\lambda}^{(m)}(\boldsymbol{x}). \tag{5.28}$$

The magenta lines in Figure 5.14 (middle, rhs) show the in-sample loss $\mathcal{L}_D^{\text{is}}$ and the estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}^{\text{blend}}, \lambda^\star)$ of this blended model. We observe that this blending blasts all other approaches in terms of model accuracy!

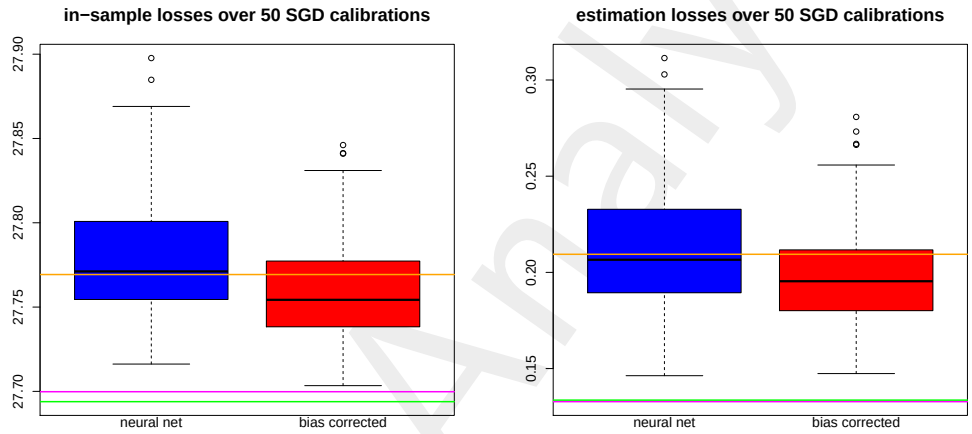


Figure 5.15: Gradient descent calibrations of the same network architecture using $M = 50$ different seeds (balance property regularized versions in red color): (lhs) in-sample losses, (rhs) estimation losses; orange line corresponds to model DNN3 of Table 5.7, magenta and green lines correspond to the blended models.

We can even go one step further. Before blending the models $\widehat{\lambda}^{(m)}(\cdot)$, $m = 1, \ldots, M$, we can apply the regularization step, i.e. the additional GLM step (5.27) to each calibration. This will generally decrease the in-sample losses and it may also act positively on the estimation losses (if we do not over-fit). This is exactly illustrated in Figure 5.15 and reflects the step from the blue box plots to the red ones. Indeed, we generally receive better models by balance property adjustments.
In the final step we can blend these regularized models to the nagging predictor analogously to (5.28). This corresponds to the green lines in Figure 5.15. We observe that for the blended models, the balance property adjustment is not necessary because the green and the magenta lines almost coincide.
We summarize our findings in Table 5.8. We observe that the blended method on line (Ch5.7) by far outperforms any other predictive model in terms of estimation loss,

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.4) CANN1	28s	703	27.9292	27.9362	0.2284	27.8940	10.1577%
(Ch5.5) CANN2	27s	780	27.9306	27.9456	0.2092	27.8684	10.2283%
(Ch5.1) DNN1	56s	1'326	28.1163	28.1216	0.3596	27.9280	10.0348%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%
(Ch5.6) DNN3 (balance)	135s	780	–	–	0.2006	27.7509	10.2691%
(Ch5.7) blended DNN	–	–	–	–	0.1336	27.6939	10.2691%

Table 5.8: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Tables 5.5 and 5.7.

i.e. compared to the true model. The in-sample loss of 27.6939 seems slightly too low, because it is lower than loss 27.7278 of the true model. This indicates (slight) over-fitting of the blended model.

In Richman–Wüthrich [111] we study convergence properties of the nagging predictor in an explicit motor third-party liability insurance example. The findings are that the nagging predictor converges in roughly 20 steps (20 blended neural network predictors) and the corresponding uncertainty becomes sufficiently small in 40 steps on portfolio level. On an individual policy level there still remains quite some uncertainty on a few policies after blending 40 models, and uncertainties only become small after blending over 400 neural network predictors. ■

Remarks 5.14.

- At latest now, we are in the two modeling cultures dilemma described by Breiman [15]. In the GLM chapter we have started off from the *data modeling culture* specifying explicit parametric models that can be analyzed and interpreted, for instance, in terms of goodness of fit. In the previous Example 5.13 we have fully arrived at the *algorithmic modeling culture* where it's all about predictive performance. That is, how can we twist and blend the models to tease a slightly better predictive accuracy.

- Our analysis has always been based on a training/learning set for fitting the models (and providing in-sample losses) and a validation set for tracking early stopping (and providing out-of-sample losses). We have been able to do our analysis based on two sets because we know the true model $\lambda^\star$ which allowed for model evaluation (in green color). If the true model is not known, one typically chooses a third data set called test data set which empirically plays the role of the true model. Thus, in that case the entire data is partitioned into three parts.

- In the GLM Chapter 2 we have also discussed AIC for model selection. AIC cannot be used within neural networks. Note that AIC is based on a bias correction that is justified by asymptotic behaviors of MLEs. In neural networks we apply early stopping, thus, we do not work with MLEs and such asymptotic results do not apply, see Section 4.2.3 in [141].

5.2 Gaussian random fields

In view of the universality theorems of Cybenko [29], Hornik et al. [69] or Leshno et al. [88], it is tempting to study the space of infinite (shallow) neural networks. These neural networks are dense in the set of compactly supported continuous functions (for non-polynomial activation functions), and hence can approximate any continuous regression function $\mu : \mathcal{X} \to \mathbb{R}$ on a compact feature space $\mathcal{X}$ arbitrarily well. We do not consider a model in full generality here, but we restrict to a Gaussian Bayesian model because in that case everything can be calculated explicitly. We follow Neal [97].

5.2.1 Gaussian Bayesian neural network

We start with a shallow neural network having $q_1 \in \mathbb{N}$ hidden neurons and we assume that the activation function ϕ is (non-polynomial and) uniformly bounded

$$\|\phi\|_\infty = \sup_{x\in\mathbb{R}} |\phi(x)| < \infty. \tag{5.29}$$

This shallow neural network provides regression function $\mu : \mathcal{X} \to \mathbb{R}$ on the q_0-dimensional feature space $\mathcal{X}$ taking the form

$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = \langle \boldsymbol{\beta}, \boldsymbol{z}(\boldsymbol{x}) \rangle,$$

where we set $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_{q_1})' \in \mathbb{R}^{q_1+1}$, $z_0(\boldsymbol{x}) \equiv 1$, and with hidden neurons

$$\boldsymbol{z}(\boldsymbol{x}) = \phi\langle W, \boldsymbol{x} \rangle \;\in\; \mathbb{R}^{q_1}.$$

Note that we use the letter μ for the regression function here because we use the linear activation for the output layer.
Next we embed this regression function into a Bayesian context by choosing a prior distribution $\pi_{q_1}(\boldsymbol{\beta}, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{q_1})$ on the parameter space $\boldsymbol{\Theta}_{q_1} = \{(\boldsymbol{\beta}, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{q_1})\}$. This choice implies that we have (prior) expected regression function

$$\boldsymbol{x} \;\mapsto\; \mathbb{E}\left[\mu(\boldsymbol{x})\right] = \mathbb{E}\left[\langle \boldsymbol{\beta}, \boldsymbol{z}(\boldsymbol{x}) \rangle\right] = \int_{\boldsymbol{\Theta}_{q_1}} \langle \boldsymbol{\beta}, \boldsymbol{z}(\boldsymbol{x}) \rangle \; d\pi_{q_1}(\boldsymbol{\beta}, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{q_1}).$$

The goal is to analyze this (Bayesian) regression function for an increasing number $q_1 \to \infty$ of hidden neurons.

Model Assumptions 5.15. *Assume uniform boundedness* (5.29) *of the activation function ϕ. For the prior distribution π_{q_1} we make the following assumptions (for given q_1):*

(1) $(\beta_j)_{j\geq 0}$ and $(\boldsymbol{w}_j)_{j\geq 1}$ are independent.

(2) The weights $\boldsymbol{w}_j$ are i.i.d. Gaussian distributed with mean $\boldsymbol{m}_w \in \mathbb{R}^{q_0+1}$ and positive definite covariance matrix $T_w \in \mathbb{R}^{(q_0+1)\times(q_0+1)}$ for $j \geq 1$.

(3) The regression parameters β_j, $j \geq 0$, are independent and $\mathcal{N}(m_j, \tau_j^2)$-distributed, with means $m_j \equiv 0$ and variances $\tau_j^2 \equiv \tau_1^2$ for $j \geq 1$.

Under the above assumptions we do not assume a uniquely identifiable parametrization, in fact, when we will let $q_1 \to \infty$, the identifiability issue will not be relevant (as we will see below). Under the above assumptions we have the following lemma.

Lemma 5.16. *Under Model Assumptions 5.15 we receive for given features $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ moment generating function in $\boldsymbol{r} = (r_1, \ldots, r_n)' \in \mathbb{R}^n$*

$$M_{\mu(\boldsymbol{X}_{1:n})}(\boldsymbol{r}) = \exp\left\{ m_0 \sum_{i=1}^n r_i + \frac{\tau_0^2}{2}\left(\sum_{i=1}^n r_i\right)^2 \right\} \mathbb{E}\left[\exp\left\{ \frac{\tau_1^2}{2}\left(\sum_{i=1}^n r_i \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \right)^2 \right\} \right]^{q_1},$$

where we define the n-dimensional random vector $\mu(\boldsymbol{X}_{1:n}) = (\mu(\boldsymbol{x}_i))'_{i=1,\ldots,n}$.

Proof of Lemma 5.16. We calculate the moment generating function of the random vector $\mu(\boldsymbol{X}_{1:n})$ in $\boldsymbol{r} \in \mathbb{R}^n$. It is given by

$$M_{\mu(\boldsymbol{X}_{1:n})}(\boldsymbol{r}) = \mathbb{E}\left[\exp\left\{\boldsymbol{r}'\mu(\boldsymbol{X}_{1:n})\right\}\right] = \mathbb{E}\left[\exp\left\{\sum_{i=1}^n r_i\left(\beta_0 + \sum_{j=1}^{q_1}\beta_j\phi\langle \boldsymbol{w}_j, \boldsymbol{x}_i\rangle\right)\right\}\right].$$

The above assumptions (1)-(3) imply that the latter decouples in j, and we obtain using normality in β_j

$$\begin{aligned}
M_{\mu(\boldsymbol{X}_{1:n})}(\boldsymbol{r}) &= \mathbb{E}\left[\exp\left\{\sum_{i=1}^n r_i\beta_0\right\}\right]\prod_{j=1}^{q_1}\mathbb{E}\left[\exp\left\{\sum_{i=1}^n r_i\beta_j\phi\langle \boldsymbol{w}_j, \boldsymbol{x}_i\rangle\right\}\right] \\
&= \mathbb{E}\left[\exp\left\{\sum_{i=1}^n r_i\beta_0\right\}\right]\prod_{j=1}^{q_1}\mathbb{E}\left[\mathbb{E}\left[\exp\left\{\sum_{i=1}^n r_i\beta_j\phi\langle \boldsymbol{w}_j, \boldsymbol{x}_i\rangle\right\}\middle| \boldsymbol{w}_j\right]\right] \\
&= \exp\left\{ m_0 \sum_{i=1}^n r_i + \frac{\tau_0^2}{2}\left(\sum_{i=1}^n r_i\right)^2 \right\}\prod_{j=1}^{q_1}\mathbb{E}\left[\exp\left\{\frac{\tau_1^2}{2}\left(\sum_{i=1}^n r_i\phi\langle \boldsymbol{w}_j, \boldsymbol{x}_i\rangle\right)^2\right\}\right].
\end{aligned}$$

The i.i.d. property of $\boldsymbol{w}_j$ provides the claim. □

5.2.2 Infinite Gaussian Bayesian neural network

We would like to consider the limit $q_1 \to \infty$ in Lemma 5.16. Note that we have assumed $\|\phi\|_\infty < \infty$. This implies

$$\log \mathbb{E}\left[\exp\left\{\frac{r^2\tau_1^2}{2}(\phi\langle \boldsymbol{w}_1, \boldsymbol{x}\rangle)^2\right\}\right] \;\leq\; \frac{r^2\tau_1^2}{2}\|\phi\|_\infty^2,$$

and we may have a similar lower bound. This suggests that we need to scale the variance $\tau_1^2 = \tau_1^2(q_1)$ as a function of the number of hidden neurons q_1 so that the system does not explode for $q_1 \to \infty$. This proposes the following extension of our model assumptions.

Model Assumptions 5.17. *In addition to Model Assumptions 5.15 we make the following assumption under prior distribution π_{q_1}:*

(4) The variance τ_1^2 of the regression parameters β_j, $j \geq 1$, is a function of the number of hidden neurons q_1 with scaling $\tau_1^2 = \tau_1^2(q_1) = \tau^2/q_1$ for some $\tau^2 > 0$.

Using Lemma 5.16 we can prove the following statement.

Proposition 5.18. *Under Model Assumptions 5.17 we receive for features $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ and $\boldsymbol{r} = (r_1, \ldots, r_n)' \in \mathbb{R}^n$*

$$\lim_{q_1 \to \infty} M_{\mu(\boldsymbol{X}_{1:n})}(\boldsymbol{r}) = \exp\left\{ m_0 \boldsymbol{r}' \boldsymbol{e}_n + \frac{1}{2} \boldsymbol{r}' C(\boldsymbol{X}_{1:n}) \boldsymbol{r} \right\},$$

with $\boldsymbol{e}_n = (1, \ldots, 1)' \in \mathbb{R}^n$ and covariance matrix

$$C(\boldsymbol{X}_{1:n}) = \left(\tau_0^2 + \tau^2 \mathbb{E}\left[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i \rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j \rangle \right] \right)_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}.$$

Sketch of proof of Proposition 5.18. The statement can be seen as follows: the assumption $\|\phi\|_\infty < \infty$ implies that the moment generating function of the random variable $\left(\sum_{i=1}^n r_i \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i \rangle \right)^2$ has a positive radius of convergence. This implies that the corresponding power series expansion exists around zero. This in turn implies that the limit $q_1 \to \infty$ can be evaluate element-wise in the power series expansion, and this provides the above result. □

The previous proposition tells us that the regression function μ converges in distribution to a Gaussian random field on $\mathcal{X}$ for $q_1 \to \infty$. The goal now is to perform Bayesian inference under the given data $\mathcal{D}$, assuming that the prior regression function is given by a Gaussian random field on the feature space $\mathcal{X}$.
For the further analysis it is going to be crucial to understand the covariance structure

$$(\boldsymbol{x}_1, \boldsymbol{x}_2)' \in \mathcal{X}^2 \quad \mapsto \quad C(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tau_0^2 + \tau^2 \mathbb{E}\left[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_1 \rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_2 \rangle \right]. \tag{5.30}$$

In Section 2.1.3, Neal [97] analyses these covariance functions for smooth activation functions ϕ and for the step function activation ϕ. The step function activation is particularly nice because it allows for a simple closed form solution in (5.30). Therefore, we will restrict to the step function activation below.

5.2.3 Bayesian inference for Gaussian random field priors

Assume that the regression function $\mu : \mathcal{X} \to \mathbb{R}$ is modeled by the Gaussian random field on $\mathcal{X}$ obtained in Proposition 5.18. For a (Gaussian) Bayesian model we make the following assumption: given realization μ, we assume that the case $(Y, \boldsymbol{x})$ is described by

$$Y|_\mu \ \sim \ \mathcal{N}(\mu(\boldsymbol{x}), \sigma^2). \tag{5.31}$$

Assume that for given μ, we have n independent cases $(Y_i, \boldsymbol{x}_i)_{i=1,\ldots,n}$ obeying (5.31). The joint density of the Gaussian random field $\mu(\boldsymbol{X}_{1:n})$ in these features $\boldsymbol{X}_{1:n} = (\boldsymbol{x}_i)_{i=1,\ldots,n}$ and the corresponding observations $\boldsymbol{Y}_{1:n} = (Y_1, \ldots, Y_n)'$ is given by

$$\begin{aligned} f\left(\boldsymbol{Y}_{1:n}, \mu(\boldsymbol{X}_{1:n})\right) \ &\propto \ \exp\left\{ -\frac{1}{2\sigma^2} \left(\boldsymbol{Y}_{1:n} - \mu(\boldsymbol{X}_{1:n})\right)' \left(\boldsymbol{Y}_{1:n} - \mu(\boldsymbol{X}_{1:n})\right) \right\} \\ &\quad \times \ \exp\left\{ -\frac{1}{2} \left(\mu(\boldsymbol{X}_{1:n}) - m_0 \boldsymbol{e}_n\right)' C(\boldsymbol{X}_{1:n})^{-1} \left(\mu(\boldsymbol{X}_{1:n}) - m_0 \boldsymbol{e}_n\right) \right\}. \end{aligned}$$

This provides the following corollary.

Corollary 5.19. *Under the above assumptions we have posterior distribution of μ in the features $\boldsymbol{X}_{1:n} = (\boldsymbol{x}_i)_{i=1,\ldots,n}$ (assuming that the covariance matrix $C(\boldsymbol{X}_{1:n})$ is positive definite) for given observations $\boldsymbol{Y}_{1:n} = (Y_1, \ldots, Y_n)'$*

$$\mu(\boldsymbol{X}_{1:n})|_{\boldsymbol{Y}_{1:n}} \sim \mathcal{N}\left(\widehat{\mu}_{1:n}, \widehat{C}_{1:n}\right),$$

with first two moments

$$\widehat{\mu}_{1:n} = \widehat{C}_{1:n}\left(\sigma^{-2}\boldsymbol{Y}_{1:n} + m_0 C(\boldsymbol{X}_{1:n})^{-1}\boldsymbol{e}_n\right) \quad \text{and} \quad \widehat{C}_{1:n} = \left(C(\boldsymbol{X}_{1:n})^{-1} + \sigma^{-2}\mathbb{1}_n\right)^{-1}.$$

Note that we need for the above corollary that $\boldsymbol{x}_i$ are mutually different for all $i = 1, \ldots, n$, otherwise the covariance matrix $C(\boldsymbol{X}_{1:n})$ will not be invertible. However, if we have more than one case with the same feature value, say $\boldsymbol{x}_1 = \boldsymbol{x}_2$, then we may consider (the sufficient statistics)

$$Y_1 + Y_2|_\mu \ \sim \ \mathcal{N}(2\mu(\boldsymbol{x}_1), 2\sigma^2),$$

and obtain a similar result. Concluding, Corollary 5.19 provides an explicit form of the posterior distribution of μ in the features $\boldsymbol{X}_{1:n}$. This corresponds to Bayesian inference *in the cases* $\boldsymbol{X}_{1:n} = (\boldsymbol{x}_i)_{i=1,\ldots,n}$.

5.2.4 Predictive distribution for Gaussian random field priors

In exactly the same fashion as above we consider the predictive distribution of random vectors $\boldsymbol{Y}_{n+1:n+k} = (Y_{n+1}, \ldots, Y_{n+k})$, for $k \geq 1$, and the corresponding mean vectors $\mu(\boldsymbol{X}_{n+1:n+k}) = (\mu(\boldsymbol{x}_i))'_{i=n+1,\ldots,n+k} \in \mathbb{R}^k$, given observations $\boldsymbol{Y}_{1:n}$.

Corollary 5.20. *Assume that the cases $(Y_1, \boldsymbol{x}_1), \ldots, (Y_{n+k}, \boldsymbol{x}_{n+k})$ are conditionally independent, given μ, having conditional distribution* (5.31)*, and μ is the Gaussian random field from above. The predictive distribution of the random vector $\boldsymbol{Y}_{n+1:n+k}$, given $\boldsymbol{Y}_{1:n}$, is multivariate Gaussian with the first two moments given by*

$$\begin{aligned}
\widehat{\mu}_{n+1:n+k|1:n} &= \mathbb{E}\left[\mu(\boldsymbol{X}_{n+1:n+k})\middle|\, \boldsymbol{Y}_{1:n}\right], \\
\widehat{C}_{n+1:n+k|1:n} &= \mathrm{Cov}\left(\mu(\boldsymbol{X}_{n+1:n+k})\middle|\, \boldsymbol{Y}_{1:n}\right) + \sigma^2\mathbb{1}_k.
\end{aligned}$$

These first two moments are obtained from the last k components of

$$\begin{aligned}
\widehat{\mu}_{1:n+k|1:n} &= E\left[\mu(\boldsymbol{X}_{1:n+k})\middle|\, \boldsymbol{Y}_{1:n}\right] \\
&= \widehat{C}_{1:n+k|1:n}\left(\sigma^{-2}(\boldsymbol{Y}'_{1:n}, 0, \ldots, 0)' + m_0 C(\boldsymbol{X}_{1:n+k})^{-1}\boldsymbol{e}_{n+k}\right), \\
\widehat{C}_{1:n+k|1:n} &= \mathrm{Cov}\left(\mu(\boldsymbol{X}_{1:n+k})\middle|\, \boldsymbol{Y}_{1:n}\right) = \left(C(\boldsymbol{X}_{1:n+k})^{-1} + \sigma^{-2}\mathbb{1}_{n+k|n}\right)^{-1},
\end{aligned}$$

with $(\boldsymbol{Y}'_{1:n}, 0, \ldots, 0)' \in \mathbb{R}^{n+k}$ and $\mathbb{1}_{n+k|n} = \mathrm{diag}(1, \ldots, 1, 0, \ldots, 0) \in \mathbb{R}^{(n+k)\times(n+k)}$ with n entries being 1 and k entries being 0.

Remark. We again assume that $C(\boldsymbol{X}_{1:n+k})$ is invertible for the above result to hold.

Proof of Corollary 5.20. We start by calculating the moment generating function for $\boldsymbol{r} \in \mathbb{R}^k$

$$\begin{aligned}
M_{\boldsymbol{Y}_{n+1:n+k}|\boldsymbol{Y}_{1:n}}(\boldsymbol{r}) &= \mathbb{E}\left[\exp\left\{\boldsymbol{r}'\boldsymbol{Y}_{n+1:n+k}\right\}\middle|\, \boldsymbol{Y}_{1:n}\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\exp\left\{\boldsymbol{r}'\boldsymbol{Y}_{n+1:n+k}\right\}\middle|\, \boldsymbol{Y}_{1:n}, \mu\right]\middle|\, \boldsymbol{Y}_{1:n}\right] \\
&= \mathbb{E}\left[\exp\left\{\boldsymbol{r}'\mu(\boldsymbol{X}_{n+1:n+k})\right\}\middle|\, \boldsymbol{Y}_{1:n}\right]\exp\left\{\frac{1}{2}\boldsymbol{r}'\boldsymbol{r}\sigma^2\right\}.
\end{aligned}$$

Thus, the claim will follow, once we have proved that $\mu(\boldsymbol{X}_{n+1:n+k})$ is multivariate Gaussian with the corresponding moments, conditionally given $\boldsymbol{Y}_{1:n}$. The latter claim follows completely similarly to Corollary 5.19. This finishes the proof. □

Remarks 5.21.

- Under the above Gaussian Bayesian assumptions the only remaining difficulties are the calculation of $G_{i,j} = \mathbb{E}[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle]$ and the inversion of (potentially) high-dimensional matrices.

- In view of the previous item we define the Gram matrix

$$\begin{aligned} G(\boldsymbol{X}_{1:n+k}) &= (G_{i,j})_{i,j=1,\ldots,n+k} \\ &= (\mathbb{E}[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle])_{i,j=1,\ldots,n+k} \in \mathbb{R}^{(n+k)\times(n+k)}. \end{aligned}$$

  In view of Corollary 5.20 this Gram matrix has three parts: (1) $G(\boldsymbol{X}_{1:n}) \in \mathbb{R}^{n\times n}$ is the in-sample Gram matrix, (2) $G(\boldsymbol{X}_{n+1:n+k}) \in \mathbb{R}^{k\times k}$ is the out-of-sample Gram matrix, and (3) $(G_{i,j})_{i=1,\ldots,n;\ j=n+1,\ldots,n+k} \in \mathbb{R}^{n\times k}$ is the cross Gram matrix.

- More general forms may be obtained under non-Gaussian assumptions: (i) the observations Y_i may have different distributional assumptions than (5.31), and also (ii) the Gaussian random field may be replaced by other random fields. The latter is discussed in Neal [97], in particular, from the point of view that different neurons in neural networks may represent different information. This interpretation has got lost in our asymptotic consideration (and therefore also identifiability can be neglected here).

5.2.5 Step function activation

We consider the step function activation $\phi(x) = \mathbb{1}_{\{x\geq 0\}}$. We have

$$G_{i,j} = \mathbb{E}[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle] = \mathbb{E}\left[\mathbb{1}_{\{\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \geq 0\}} \mathbb{1}_{\{\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle \geq 0\}}\right].$$

The crucial size is the angle between $\boldsymbol{x}_i \in \mathbb{R}^{q_0+1}$ and $\boldsymbol{x}_j \in \mathbb{R}^{q_0+1}$ (we include the intercepts $x_{i,0} = x_{j,0} = 1$ in $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively, here). This angle is given by

$$\gamma = \arccos\left(\frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}\right) \in [0, \pi].$$

Model Assumptions 5.22. *In addition to Model Assumptions 5.17 we assume that the distribution of $\boldsymbol{w}_1$ is rotationally invariant.*

Under this additional assumption we obtain

$$G_{i,j} = \mathbb{E}[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle] = \frac{1}{2} - \frac{1}{2\pi}\arccos\left(\frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}\right) \in [0, 1/2].$$

In fact, the upper bound is obtained if and only if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ point in the same direction (in the $\mathbb{R}^{q_0+1}$ space). Thus, under the step function activation $\phi(x) = \mathbb{1}_{\{x\geq 0\}}$ and under rotational invariance of $\boldsymbol{w}_1$ we obtain covariance function

$$\boldsymbol{X}_{1:n} \;\mapsto\; C(\boldsymbol{X}_{1:n}) = \left(\tau_0^2 + \frac{\tau^2}{2} - \frac{\tau^2}{2\pi}\arccos\left(\frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}\right)\right)_{i,j=1,\ldots,n} \in \mathbb{R}^{n\times n}.$$

We study the sensitivity of this covariance function for small perturbations of the feature values. Therefore, we consider features $\boldsymbol{x} = \boldsymbol{x}_i$ and $\boldsymbol{x}_j = \boldsymbol{x} + \varepsilon(0, \boldsymbol{z}')'$ for $\varepsilon \in \mathbb{R}$ and $\boldsymbol{z} \in \mathbb{R}^{q_0}$ with $\|\boldsymbol{z}\| = 1$. In view of the previous identity we have for this perturbation

$$\begin{aligned}\varepsilon \;\mapsto\; g_{\boldsymbol{x}}(\varepsilon) &\overset{\text{def.}}{=} \mathbb{E}\left[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}_i\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x}_j\rangle\right] \;=\; \mathbb{E}\left[\phi\langle \boldsymbol{w}_1, \boldsymbol{x}\rangle \phi\langle \boldsymbol{w}_1, \boldsymbol{x} + \varepsilon(0, \boldsymbol{z}')'\rangle\right] \\ &= \frac{1}{2} - \frac{1}{2\pi}\arccos\left(\frac{\|\boldsymbol{x}\|^2 + \varepsilon\langle \boldsymbol{x}, \boldsymbol{z}\rangle}{\|\boldsymbol{x}\|\sqrt{\|\boldsymbol{x}\|^2 + 2\varepsilon\langle \boldsymbol{x}, \boldsymbol{z}\rangle + \varepsilon^2}}\right). \qquad (5.32)\end{aligned}$$

Note that we have $g(0) = 1/2$. We abbreviate $z = \langle \boldsymbol{x}, \boldsymbol{z}\rangle$. The Cauchy-Schwarz inequality implies, we use $\|\boldsymbol{z}\| = 1$,

$$z^2 = \langle \boldsymbol{x}, \boldsymbol{z}\rangle^2 = \left(\sum_{l=1}^{q_0} x_l z_l\right)^2 \leq \sum_{l=1}^{q_0} x_l^2 < 1 + \sum_{l=1}^{q_0} x_l^2 = \|\boldsymbol{x}\|^2,$$

where the latter square norm involves the intercept $x_0 = 1$. This implies that $\|\boldsymbol{x}\|^2 - z^2 > 0$. We define

$$x(\varepsilon) = \frac{\|\boldsymbol{x}\|^2 + \varepsilon z}{\|\boldsymbol{x}\|\sqrt{\|\boldsymbol{x}\|^2 + 2\varepsilon z + \varepsilon^2}}.$$

We first calculate the derivative of $\varepsilon \mapsto x(\varepsilon)$. It is given by

$$\frac{\partial}{\partial\varepsilon}x(\varepsilon) = \frac{z\left(\|\boldsymbol{x}\|^2 + 2\varepsilon z + \varepsilon^2\right) - \left(\|\boldsymbol{x}\|^2 + \varepsilon z\right)(z + \varepsilon)}{\|\boldsymbol{x}\|\left(\|\boldsymbol{x}\|^2 + 2\varepsilon z + \varepsilon^2\right)^{3/2}} = -\frac{\varepsilon\left(\|\boldsymbol{x}\|^2 - z^2\right)}{\|\boldsymbol{x}\|\left(\|\boldsymbol{x}\|^2 + 2\varepsilon z + \varepsilon^2\right)^{3/2}}.$$

We conclude that

$$\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}x(\varepsilon) = \lim_{\varepsilon\to 0} -\frac{\varepsilon\left(\|\boldsymbol{x}\|^2 - z^2\right)}{\|\boldsymbol{x}\|\left(\|\boldsymbol{x}\|^2 + 2\varepsilon z + \varepsilon^2\right)^{3/2}} = 0.$$

We now come back to function (5.32) given by

$$\varepsilon \;\mapsto\; g_{\boldsymbol{x}}(\varepsilon) = \frac{1}{2} - \frac{1}{2\pi}\arccos\left(x(\varepsilon)\right).$$

Its derivative for $\varepsilon \neq 0$ is given by

$$\frac{\partial}{\partial\varepsilon}g_{\boldsymbol{x}}(\varepsilon) = \frac{1}{2\pi}\frac{x'(\varepsilon)}{\sqrt{1 - x(\varepsilon)^2}}.$$

Observe that both enumerator and denominator of the last ratio converge to zero as $\varepsilon \to 0$. Therefore, we can apply l'Hôpital's rule to obtain, we also use $\|\boldsymbol{x}\|^2 - z^2 > 0$,

$$\begin{aligned}\lim_{\varepsilon\to 0}\frac{\partial}{\partial\varepsilon}g_{\boldsymbol{x}}(\varepsilon) &= \frac{1}{2\pi}\lim_{\varepsilon\to 0}\frac{x'(\varepsilon)}{\sqrt{1 - x(\varepsilon)^2}} \;=\; -\frac{1}{2\pi}\left(\lim_{\varepsilon\to 0}\frac{(x'(\varepsilon))^2}{1 - x(\varepsilon)^2}\right)^{1/2} \\ &= -\frac{1}{2\pi}\left(\lim_{\varepsilon\to 0}\frac{x''(\varepsilon)}{-x(\varepsilon)}\right)^{1/2} = -\frac{1}{2\pi}\|\boldsymbol{x}\|^{-2}\sqrt{\|\boldsymbol{x}\|^2 - z^2} \;<\; 0.\end{aligned}$$

From this we conclude that the function (5.32) is approximately linear for small $\varepsilon > 0$ with negative slope $-(2\pi)^{-1}\|\boldsymbol{x}\|^{-2}\sqrt{\|\boldsymbol{x}\|^2 - z^2}$. Thus, locally the Gaussian random field behaves as a Brownian motion.

Remarks. The case for the step function activation $\phi(x) = \mathbb{1}_{\{x \geq 0\}}$ and rotationally invariant weights $\boldsymbol{w}_1$ (according to Model Assumptions 5.22) is completely solved. Note that through definition (5.30) of the covariance function $C(\boldsymbol{X}_{1:n})$ we still keep track of the original shallow neural network model with $q_1 \to \infty$ hidden neurons. In general, however, this is not necessary and we could directly start with a Bayesian random field prior model by assuming that

- the regression function μ is a random field on the feature space $\mathcal{X}$; and
- the cases $(Y, \boldsymbol{x})$ are independent having conditional distribution

$$Y|_{\mu} \;\sim\; F_{\mu(\boldsymbol{x})},$$

  with $\mu(\boldsymbol{x})$ characterizing the conditional mean of Y with feature $\boldsymbol{x} \in \mathcal{X}$, given μ.

If we start from such a random field prior model we should ensure that μ is comparably smooth. Otherwise interpretation of insurance prices is not meaningful in the sense that neighboring features receive similar prices.
There are many more examples of analytically tractable Gram matrices. For instance, if the activation function ϕ is a cumulative distribution function, then the Gram matrix is analytically tractable. This has been explored in Section 2.2 of Xiang [142].

Chapter 6

Classification and Regression Trees

In this chapter we present the classification and regression tree (CART) technique introduced and studied in Breiman et al. [16]. We closely follow this reference, but we use a modified notation. For our examples we use the R package `rpart`; as mentioned in Therneau–Atkinson [125], CART is a trademarked software package, therefore the routine in R is called `rpart` which comes from `r`ecursive `part`itioning.

6.1 Binary Poisson regression trees

Assume that the cases $(N_i, \boldsymbol{x}_i, v_i)$ are independent and N_i are Poisson distributed for $i = 1, \ldots, n$ with expected claims counts $\mathbb{E}[N_i] = \lambda(\boldsymbol{x}_i)v_i$ for a given regression function $\lambda : \mathcal{X} \to \mathbb{R}_+$. In Chapter 2 on GLMs we have made log-linear assumption for the modeling of the expected frequency $\boldsymbol{x} \mapsto \lambda(\boldsymbol{x})$, see (2.2). This GLM approach assumes a fixed structural form of the expected frequency $\lambda(\cdot)$. Similarly to neural networks, regression trees are more general in the sense that they try to learn an optimal structural form from the data $\mathcal{D}$. The idea is to design an algorithm that partitions the feature space $\mathcal{X}$ into disjoint (homogeneous)[1] subsets $(\mathcal{X}_t)_{t\in\mathcal{T}}$, where $\mathcal{T}$ is a finite set of indexes. On each subset $\mathcal{X}_t$ we choose a frequency parameter $\bar{\lambda}_t$ that describes the expected frequency on that (homogeneous) part $\mathcal{X}_t$ of the feature space $\mathcal{X}$. Finally, we estimate the (unknown) expected frequency on the total feature space $\mathcal{X}$ by

$$\boldsymbol{x} \;\mapsto\; \widehat{\lambda}(\boldsymbol{x}) \;=\; \sum_{t\in\mathcal{T}} \bar{\lambda}_t \, \mathbb{1}_{\{\boldsymbol{x}\in\mathcal{X}_t\}}. \tag{6.1}$$

There are two things we need to explain: (i) the choice of a 'good' partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of the feature space $\mathcal{X}$ and (ii) the choices of the frequency parameters $\bar{\lambda}_t$ for each $t \in \mathcal{T}$.

Remark that this partitioning into homogeneous subsets is rather similar to categorical coding of continuous variables in GLMs when they do not meet the required functional form, for instance, the monotonicity assumption.

[1] The meaning of homogeneity will become clear below.

6.1.1 Binary trees and binary indexes

For the partitioning we use a *binary tree* growing algorithm. This binary tree growing algorithm starts by partitioning the feature space $\mathcal{X}$ into two disjoint subsets $\mathcal{X}_0$ and $\mathcal{X}_1$; these subsets may then further be partitioned into two disjoint subsets, and so forth. To describe this partitioning process we introduce a binary tree notation.

The binary tree growing algorithm can be described recursively as follows.

- We initialize *(root) generation* $k = 0$. We define the initial binary index $t = 0$ and we set $\mathcal{X}_0 = \mathcal{X}$. This is the *root* or *root node* of the binary tree.
- Assume $\mathcal{X}_t$ is a *node of generation* k with binary index t. This node can be partitioned into two *nodes of generation* $k+1$ denoted by $\mathcal{X}_{t0}$ and $\mathcal{X}_{t1}$. Node $\mathcal{X}_{t0}$ is called the *left child* and node $\mathcal{X}_{t1}$ the *right child* of the *mother* node $\mathcal{X}_t$. These children have binary indexes in $\{0\} \times \{0,1\}^{k+1}$. Note that indexes $t0$ and $t1$ are always understood in the concatenation sense.

  Choose, say, the node $\mathcal{X}_t$ in generation $k = 3$ with binary index $t = 0101$. This node is the right child of the mother node with index 010 (belonging to generation 2) and the potential children have binary indexes 01010 and 01011 (in generation 4). Thus, binary indexes (chosen in a consistent way) describe a tree and the length of the binary index always indicates to which generation this particular node is belonging to. Moreover, it allows us to identify the relationship between generations.

  We denote a *binary split* of node $\mathcal{X}_t$ by $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$. A binary split $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ is always assumed to be a partition of $\mathcal{X}_t$, that is, $\mathcal{X}_{t0} \cup \mathcal{X}_{t1} = \mathcal{X}_t$ and $\mathcal{X}_{t0} \cap \mathcal{X}_{t1} = \emptyset$.
- After applying a stopping rule to the binary tree growing algorithm we obtain a set of binary indexes T that label the resulting nodes $(\mathcal{X}_t)_{t\in\mathsf{T}}$. This set T describes the structure of a binary tree which allows us to identify generations and relationships.

  All nodes that do *not* have children are called *leaves* of the binary tree and are indexed by $\mathcal{T} \subset \mathsf{T}$. The leaves $(\mathcal{X}_t)_{t\in\mathcal{T}}$ provide a partition of $\mathcal{X}$.

Binary Tree Growing Algorithm.

---

(0) Initialize $\mathcal{X}_0 = \mathcal{X}$ and $\mathsf{T} = \mathcal{T} = \{0\}$.

(1) Repeat until a stopping rule is exercised: select $t \in \mathcal{T}$ and the corresponding $\mathcal{X}_t$,

  (a) apply a binary split $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ to $\mathcal{X}_t$;

  (b) return the new binary tree $(\mathcal{X}_t)_{t\in\mathsf{T}}$ with indexes

$$\begin{aligned} \mathsf{T} &\leftarrow \mathsf{T} \cup \{t0, t1\}, \\ \mathcal{T} &\leftarrow (\mathcal{T} \setminus \{t\}) \cup \{t0, t1\}. \end{aligned}$$

(2) Return the binary tree $(\mathcal{X}_t)_{t\in\mathsf{T}}$ with binary indexes T, leaf indexes $\mathcal{T}$ and partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of $\mathcal{X}$.

A 'good' partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of the feature space $\mathcal{X}$ for our prediction problem will describe good choices of binary splits $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of nodes $\mathcal{X}_t$, and also a stopping rule when no further binary split should be applied to a particular node and binary tree, respectively.

6.1.2 Pre-processing features: standardized binary splits

In step (1a) of the Binary Tree Growing Algorithm we may consider any possible (non-trivial) binary split $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$. Typically, this leads to too much complexity and computational burden. Therefore, we only consider so-called (non-trivial) *standardized binary splits* (SBS). Remark that these non-trivial SBS do *not* need pre-processing of features, i.e. this is different from the GLMs and neural networks considered above.
The general set-up is that we have a q-dimensional feature space $\mathcal{X}$ and we want to estimate a regression function $\lambda(\cdot) : \mathcal{X} \to \mathbb{R}_+$ with function (6.1). For this goal, we only consider SBS of nodes $\mathcal{X}_t$ in step (1a) of the Binary Tree Growing Algorithm; these SBS lead to 'rectangles' on $\mathcal{X}$ and are described as follows.

Standardized Binary Split.

A standardized binary split (SBS) $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ is done as follows:

1. consider only a single component x_l of the feature $\boldsymbol{x} = (x_1, \ldots, x_q) \in \mathcal{X}_t$ at a time;
2. for ordered components x_l we ask split questions: $x_l \leq c$, for given values c;
3. for nominal components x_l we ask split questions: $x_l \in C$, for non-empty subsets C of the possible labels of the feature component x_l.

Since we only have finitely many cases $(N_i, \boldsymbol{x}_i, v_i)$ in the data $\mathcal{D}$, there exist at most finitely many non-trivial SBS questions until the observed (different) features $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are fully atomized by the Binary Tree Growing Algorithm (if we assume that the separation lines always lie at half distance between continuous feature components). In the sequel we always exclude trivial SBS, where trivial SBS in this context means that either $\mathcal{X}_{t0}$ or $\mathcal{X}_{t1}$ does not contain at least one observed feature $\boldsymbol{x}_i$ of the data $\mathcal{D}$. This is a standard assumption that we will not mention each time below, but tacitly assume.

6.1.3 Goodness of split

To apply the Binary Tree Growing Algorithm with SBS we still need to explain how we select $t \in \mathcal{T}$ in step (1) and which SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ we apply to $\mathcal{X}_t$ in step (1a). This selection is done by a goodness of split criterion. Similarly to above we use the *Poisson deviance loss* as our objective function (under the assumption of having independent Poisson distributed samples).

Optimal splits for continuous feature components

Assume for the moment that all feature components of $\boldsymbol{x} = (x_1, \ldots, x_q) \in \mathcal{X}$ are continuous and we are searching for the optimal SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ for given data $\mathcal{D}$ and given $t \in \mathcal{T}$. Thus, we are looking for an optimal feature component x_l of $\boldsymbol{x}$ and an optimal constant $c \in \mathbb{R}$ such that the resulting Poisson deviance loss is minimized:

$$\min_{1\leq l\leq q,\ c\in\mathbb{R}} \left[\min_{\lambda_0\geq 0} \sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_t,\ x_{i,l}\leq c} D^*\left(N_i, \lambda_0\right) + \min_{\lambda_1\geq 0} \sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_t,\ x_{i,l}>c} D^*\left(N_i, \lambda_1\right) \right], \tag{6.2}$$

where $D^*\left(N_i, \lambda\right)$ is the Poisson deviance loss of (single) case $(N_i, \boldsymbol{x}_i, v_i)$ for expected frequency $\lambda \geq 0$, see (1.5). As mentioned above, we only consider non-trivial SBS, which implies that we only consider SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ on non-empty sets $\{\boldsymbol{x}_i \in \mathcal{X}_t :\ x_{i,l} \leq c\}$ for the left child $\mathcal{X}_{t0}$ and $\{\boldsymbol{x}_i \in \mathcal{X}_t :\ x_{i,l} > c\}$ for the right child $\mathcal{X}_{t1}$ of $\mathcal{X}_t$. Moreover, for continuous feature components x_l the constant c is chosen at half distance between the feature components $x_{i,l}$ of "neighboring observations". The inner optimizations in (6.2) can then be solved explicitly and provide the MLEs for the SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$, set $\tau = 0, 1$,

$$\bar{\lambda}_{t\tau} \ = \ \operatorname*{argmin}_{\lambda_\tau\geq 0} \sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_{t\tau}} D^*\left(N_i, \lambda_\tau\right) \ = \ \frac{\sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_{t\tau}} N_i}{\sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_{t\tau}} v_i}. \tag{6.3}$$

This allows us to rewrite the SBS optimization problem (6.2) as follows:

Determine the optimal (non-trivial) SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ solving

$$\min_{\text{SBS } \varsigma_t=(\mathcal{X}_{t0},\mathcal{X}_{t1})} \ D^*_{\mathcal{X}_{t0}}\left(\boldsymbol{N}, \bar{\lambda}_{t0}\right) + D^*_{\mathcal{X}_{t1}}\left(\boldsymbol{N}, \bar{\lambda}_{t1}\right), \tag{6.4}$$

where $\bar{\lambda}_{t\tau}$ is the MLE on $\mathcal{X}_{t\tau}$ given by (6.3) and the Poisson deviance loss on $\mathcal{X}_{t\tau}$, $\tau = 0, 1$, is given by

$$D^*_{\mathcal{X}_{t\tau}}\left(\boldsymbol{N}, \bar{\lambda}_{t\tau}\right) \ = \ \sum_{i:\ \boldsymbol{x}_i\in\mathcal{X}_{t\tau}} 2N_i \left[\frac{\bar{\lambda}_{t\tau} v_i}{N_i} - 1 - \log\left(\frac{\bar{\lambda}_{t\tau} v_i}{N_i} \right) \right],$$

where the right-hand side is set equal to $2\bar{\lambda}_{t\tau} v_i$ if $N_i = 0$.

Optimization problem (6.4) describes an in-sample Poisson deviance loss minimization.

Processing features: categorical feature components

In the above optimization (6.2) we have assumed that all feature components x_l are continuous (or at least ordered) which motivates the choice of the splitting values c. For a nominal categorical feature component x_l, the split criterion c may be replaced by non-empty subsets C that describe the possible labels of the categorical feature component x_l, see Section 6.1.2. In particular, features do not need pre-processing to apply this SBS tree growing algorithm. However, this evaluation of categorical feature labels may be computationally too expensive. For instance, if we aim at partitioning the criterion Swiss canton (there are 26 cantons in Switzerland) we obtain $2^{26-1} - 1 = 33'554'431$ possible SBS. For this reason, a categorical feature component x_l is often replaced by an ordered version thereof for the SBS. In the Poisson case this is done as follows: calculate

for each categorical label of x_l on $\mathcal{X}_t$ the empirical frequency $\bar{\lambda}_t(x_l)$ (on $\mathcal{X}_t$) and use these empirical frequencies $\bar{x}_l = \bar{\lambda}_t(x_l)$ as replacements for the categorical feature component x_l for an ordered SBS of $\mathcal{X}_t$ w.r.t. $\bar{x}_l$.
This latter approach only works well as long as $\mathcal{X}_t$ contains sufficiently many cases $(N_i, \boldsymbol{x}_i, v_i)$. If the number of cases in $\mathcal{X}_t$ is too small, then it is likely that we do not observe sufficiently many claims for certain categorical feature labels, in particular, in the low frequency case, and thus the empirical frequencies do not provide a useful ordering.

Bayesian approach

For given data $\mathcal{D}$ and a given node $\mathcal{X}_t \subset \mathcal{X}$ we can find the best SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ w.r.t. (6.4). A general issue that often occurs in insurance claims frequency modeling is that on certain nodes $\mathcal{X}_{t\tau}$ we may get a MLE $\bar{\lambda}_{t\tau}$ that is equal to zero, see (6.3). This happens in particular when the expected frequency and the overall volume on $\mathcal{X}_{t\tau}$ are small. The overall volume on $\mathcal{X}_{t\tau}$ is defined by (we always assume that there is at least one case $(N_i, \boldsymbol{x}_i, v_i)$ in the considered part of the feature space)

$$w_{t\tau} = \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_{t\tau}} v_i > 0, \tag{6.5}$$

and the expected frequency for data $\mathcal{D}$ on $\mathcal{X}_{t\tau}$ is given by

$$\lambda_{t\tau} = \frac{1}{w_{t\tau}} \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_{t\tau}} v_i\ \lambda(\boldsymbol{x}_i) > 0, \tag{6.6}$$

if we assume that the true model has expected frequency function $\boldsymbol{x} \mapsto \lambda(\boldsymbol{x})$. Thus, if the MLE $\bar{\lambda}_{t\tau} = 0$ for $\lambda_{t\tau} > 0$, we obtain a degenerate (calibrated) Poisson distribution on $\mathcal{X}_{t\tau}$ which, of course, is nonsense from a practical point of view. For this reason, the MLE $\bar{\lambda}_{t\tau}$ on $\mathcal{X}_{t\tau}$ is replaced by a Bayesian estimator $\bar{\lambda}^{\text{post}}_{t\tau}$.

Model Assumptions 6.1. *Assume we are given a portfolio $(\boldsymbol{x}_i, v_i)_{i=1,\ldots,n}$ and a regression function $\lambda(\cdot) : \mathcal{X} \to \mathbb{R}_+$. Choose $\Theta \sim \Gamma(\gamma, \gamma)$ and assume that conditionally, given Θ, the random variables $N_1, \ldots, N_n$ are independent with conditional distribution*

$$N_i|_\Theta \ \sim\ \text{Poi}\left(\lambda(\boldsymbol{x}_i)\Theta v_i\right),$$

for $i = 1, \ldots, n$.

The above assumptions introduce uncertainty in the expected frequency using a multiplicative perturbation Θ of λ. The coefficient of variation of the (expected) frequency $\lambda(\boldsymbol{x}_i)\Theta$ is

$$\text{Vco}\left(\lambda(\boldsymbol{x}_i)\Theta\right) = \text{Vco}\left(\Theta\right) = \gamma^{-1/2},$$

that is, we have a coefficient of variation that does not depend on the features.

Lemma 6.2. *Set Model Assumptions 6.1 and choose $\mathcal{X}_{t\tau} \subset \mathcal{X}$ such that $\boldsymbol{x}_i \in \mathcal{X}_{t\tau}$ for at least one $i = 1, \ldots, n$. Conditionally, given Θ, we have*

$$N_{t\tau} = \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_{t\tau}} N_i\ |_\Theta \ \sim\ \text{Poi}\left(\lambda_{t\tau}\Theta w_{t\tau}\right),$$

with $\lambda_{t\tau}$ given by (6.6) *and $w_{t\tau}$ given by* (6.5)*. The posterior expected frequency on $\mathcal{X}_{t\tau}$, given observation $N_{t\tau}$, is given by*

$$\mathbb{E}\left[\lambda_{t\tau}\Theta\middle|\, N_{t\tau}\right] = \lambda_{t\tau}\mathbb{E}\left[\Theta\middle|\, N_{t\tau}\right] = \alpha_{t\tau}\ \bar{\lambda}_{t\tau} + (1-\alpha_{t\tau})\ \lambda_{t\tau},$$

with MLE $\bar{\lambda}_{t\tau}$ given by (6.3) *and with credibility weight*

$$\alpha_{t\tau} = \frac{w_{t\tau}\lambda_{t\tau}}{\gamma + w_{t\tau}\lambda_{t\tau}} \in (0,1).$$

Proof of Lemma 6.2. The first statement immediately follows from Lemma 1.3, see also Remarks 4.3. For the second statement we define $\Lambda_{t\tau} = \lambda_{t\tau}\Theta$; $\Lambda_{t\tau}$ has a gamma distribution with shape parameter γ and scale parameter $\gamma/\lambda_{t\tau}$, see formula (3.5) in Wüthrich [135]. Therefore, the assumptions of Corollary 4.4 are fulfilled and we obtain the claim. □

Remarks 6.3.

- For given prior mean $\lambda_{t\tau} > 0$ and given shape parameter $\gamma > 0$ we can calculate the posterior mean $\mathbb{E}\left[\lambda_{t\tau}\Theta\middle|\, N_{t\tau}\right]$ as an estimator for the expected frequency on $\mathcal{X}_{t\tau}$. This posterior mean is always strictly positive and therefore does not have the same deficiency as the MLE $\bar{\lambda}_{t\tau}$ given by (6.3).
- The difficulty in practice is that the prior mean $\lambda_{t\tau}$ is not known and needs to be replaced by another estimator. The natural choices are the MLE $\bar{\lambda}_t$ on $\mathcal{X}_t$ or the Bayesian estimator $\mathbb{E}\left[\lambda_t\Theta\middle|\, N_t\right]$ on $\mathcal{X}_t$. The former has the disadvantage that it does not solve the problem of positivity and the latter needs to be calculated recursively. Thus, for the latter we may define recursively

$$\bar{\lambda}_{t\tau}^{\text{post}} = \widehat{\mathbb{E}}\left[\lambda_{t\tau}\Theta\middle|\, N_{t\tau}\right] = \widehat{\alpha}_{t\tau}\ \bar{\lambda}_{t\tau} + (1-\widehat{\alpha}_{t\tau})\ \bar{\lambda}_t^{\text{post}}, \tag{6.7}$$

  with estimated credibility weight

$$\widehat{\alpha}_{t\tau} = \frac{w_{t\tau}\bar{\lambda}_t^{\text{post}}}{\gamma + w_{t\tau}\bar{\lambda}_t^{\text{post}}} \ \in\ (0,1),$$

  and initialization $\bar{\lambda}_0^{\text{post}} = \bar{\lambda}_0 = \sum_{i=1}^n N_i / \sum_{i=1}^n v_i$ which is the (homogeneous) overall MLE for $\lambda_0 = \sum_{i=1}^n v_i\ \lambda(\boldsymbol{x}_i)/\sum_{i=1}^n v_i$.

  - The credibility weights $\alpha_{t\tau}$ and $\widehat{\alpha}_{t\tau}$ are increasing in $\lambda_{t\tau}$ and $\bar{\lambda}_t^{\text{post}}$, respectively. Therefore, good risks $\lambda_{t\tau} < \bar{\lambda}_t^{\text{post}}$ obtain a higher credibility weight under (6.7) and bad risks a lower credibility weight for the observations on $\mathcal{X}_{t\tau}$.
  - Replacing $\lambda_{t\tau}$ by $\bar{\lambda}_t^{\text{post}}$ shrinks the prior mean of the credibility estimator towards the overall estimate $\bar{\lambda}_t^{\text{post}}$ on $\mathcal{X}_t$ and diminishes structural differences.
  - Of course, this is non-optimal, but an improved version would require more knowledge about the tree structure. Such tree structure knowledge would also allow us to replace Model Assumptions 6.1 by individual assumptions on each leaf of the tree, for instance, each leaf $t \in \mathcal{T}$ may have its own (independent) Θ_t. Note that Model Assumptions 6.1 also introduce (undesirable) dependencies between the leaves through Θ.

– The remaining parameter is the coefficient of variation $\gamma^{-1/2}$. This coefficient of variation is an input parameter in the R package `rpart`, see Therneau–Atkinson [125], and needs to be chosen externally. It should not be chosen too small so that it allows us to model a good range of possible expected frequencies $\lambda(\boldsymbol{x}_i)$ which may be of magnitude 4 between good and bad risks.
– A more thorough treatment would embed (6.7) into a hierarchical credibility model, see Bühlmann–Gisler [18].

The R command `rpart` (used for the examples below) uses a less sophisticated approach than (6.7), namely, it replaces $\bar{\lambda}_{t\tau}^{\text{post}}$ by

$$\bar{\lambda}_{t\tau}^{\text{B}} = \widehat{\alpha}_{t\tau}^{\text{B}}\ \bar{\lambda}_{t\tau} + (1 - \widehat{\alpha}_{t\tau}^{\text{B}})\ \bar{\lambda}_0, \tag{6.8}$$

with (homogeneous) overall MLE $\bar{\lambda}_0$ as prior mean and with estimated credibility weights

$$\widehat{\alpha}_{t\tau}^{\text{B}} = \frac{w_{t\tau}\bar{\lambda}_0}{\gamma + w_{t\tau}\bar{\lambda}_0}\ \in\ (0,1), \tag{6.9}$$

and γ is chosen exogenously. In our examples we will use this approach (6.8)-(6.9).

6.1.4 Standardized binary split tree growing algorithm

We come back to the Binary Tree Growing Algorithm introduced on page 146. In the previous two sections we have discussed how to choose an optimal SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ for a given binary index $t \in \mathcal{T}$ using the goodness of split criterion (6.4). In this section we discuss the choice of an optimal binary index $t \in \mathcal{T}$ to perform the SBS.

The algorithm and Poisson deviance losses

We apply SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ to grow a binary tree. The aim at each iteration step is to choose the partition in (6.4) that maximizes the increase in goodness of split, measured by the decrease of the Poisson deviance loss. We define the decrease in Poisson deviance loss of a binary split $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ by

$$\Delta D^*_{\mathcal{X}_t}(\varsigma_t) = D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) - \left[D^*_{\mathcal{X}_{t0}}\left(\boldsymbol{N}, \bar{\lambda}_{t0}\right) + D^*_{\mathcal{X}_{t1}}\left(\boldsymbol{N}, \bar{\lambda}_{t1}\right)\right]. \tag{6.10}$$

Lemma 6.4. *For any binary split $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ we have $\Delta D^*_{\mathcal{X}_t}(\varsigma_t) \geq 0$.*

Proof of Lemma 6.4. Observe that we have the following inequality

$$\begin{aligned} D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) &= \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_t} 2N_i \left[\frac{\bar{\lambda}_t v_i}{N_i} - 1 - \log\left(\frac{\bar{\lambda}_t v_i}{N_i}\right)\right] \\ &= \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_{t0}} 2N_i \left[\frac{\bar{\lambda}_t v_i}{N_i} - 1 - \log\left(\frac{\bar{\lambda}_t v_i}{N_i}\right)\right] + \sum_{i:\ \boldsymbol{x}_i \in \mathcal{X}_{t1}} 2N_i \left[\frac{\bar{\lambda}_t v_i}{N_i} - 1 - \log\left(\frac{\bar{\lambda}_t v_i}{N_i}\right)\right] \\ &\geq D^*_{\mathcal{X}_{t0}}\left(\boldsymbol{N}, \bar{\lambda}_{t0}\right) + D^*_{\mathcal{X}_{t1}}\left(\boldsymbol{N}, \bar{\lambda}_{t1}\right). \end{aligned}$$

The latter follows because the MLEs minimize the corresponding Poisson deviance losses. □

The above motivates the following binary tree growing algorithm.

SBS Tree Growing Algorithm.

---

(0) Initialize $\mathcal{X}_0 = \mathcal{X}$ and $\mathsf{T} = \mathcal{T} = \{0\}$.

(1) Repeat until a stopping rule is exercised:

(a) calculate the optimal SBS on the leaves indexed by $\mathcal{T}$

$$(t, \varsigma_t) = \operatorname*{argmax}_{s \in \mathcal{T},\ \varsigma_s = (\mathcal{X}_{s0}, \mathcal{X}_{s1})} \Delta D^*_{\mathcal{X}_s}(\varsigma_s) \ \geq\ 0, \tag{6.11}$$

with a deterministic rule if there is more than one optimal SBS;

(b) if $\Delta D^*_{\mathcal{X}_t}(\varsigma_t) > 0$, return the new binary tree $(\mathcal{X}_t)_{t \in \mathsf{T}}$ with indexes

$$\begin{aligned} \mathsf{T} &\leftarrow \mathsf{T} \cup \{t0, t1\}, \\ \mathcal{T} &\leftarrow (\mathcal{T} \setminus \{t\}) \cup \{t0, t1\}; \end{aligned}$$

(c) decide whether the algorithm should be stopped and go to step (2), or otherwise return to step (1a).

(2) Return the binary tree $(\mathcal{X}_t)_{t \in \mathsf{T}}$ with binary indexes T, leaf indexes $\mathcal{T}$ and partition $(\mathcal{X}_t)_{t \in \mathcal{T}}$ of $\mathcal{X}$.

Remarks.

- The above algorithm always chooses the leaf index $t \in \mathcal{T}$ which leads to the (locally) biggest improvement in the corresponding Poisson deviance loss. In view of step (1b) we only consider SBS that lead to a real improvement $\Delta D^*_{\mathcal{X}_t}(\varsigma_t) > 0$, otherwise the algorithm should be terminated.

- Often the maximization in (6.11) is more restricted, for instance, we may restrict to SBS $\varsigma_s = (\mathcal{X}_{s0}, \mathcal{X}_{s1})$ of $\mathcal{X}_s$ which lead to new leaves $\mathcal{X}_{s0}$ and $\mathcal{X}_{s1}$ that contain at least a minimal number of cases; this is an input parameter in `rpart`.

- The only open point is the choice of a sensible stopping rule, for instance, the improvement in Poisson deviance loss in (6.11) should exceed a certain (relative) threshold, otherwise the algorithm is terminated; this is an input parameter in `rpart`.

- Note that the above algorithm could also be considered for other objective functions to which a lemma similar to Lemma 6.4 applies.

Stopping rule

The Poisson deviance loss on the partition $(\mathcal{X}_t)_{t \in \mathcal{T}}$ of $\mathcal{X}$ is given by

$$D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t \in \mathcal{T}}) \ =\ \sum_{t \in \mathcal{T}} D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right). \tag{6.12}$$

Completely analogously to Lemma 6.4 we have the following statement.

Corollary 6.5. *Consider for a partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of $\mathcal{X}$, and an additional SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ for a given $t \in \mathcal{T}$. This gives the new leaf indexes $\mathcal{T}' = (\mathcal{T}\setminus\{t\})\cup\{t0, t1\}$. We have $D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}}) \geq D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}'})$.*

Proof. The proof is a straightforward consequence of Lemma 6.4. □

Corollary 6.5 says that every extra SBS decreases the objective function $D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}})$ and a sensible stopping rule would explore the amount of decrease of the Poisson deviance loss in relation to the number of splits (and parameters $\bar\lambda_s$, $s \in \mathcal{T}$) involved. This is quite similar to the likelihood ratio test (2.15). However, in many situations this is not a feasible way and the problem is rather solved by growing a very large tree in a first step, and in a second step this large tree is pruned by splits that do not contribute sufficiently to a modeling improvement. Details are presented in Section 6.2 on pruning.

Assignment rule and regression function

The SBS Tree Growing Algorithm on page 151 establishes us with a partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of $\mathcal{X}$ having leaf indexes $\mathcal{T}$. This provides the *regression tree estimator* for the expected frequency $\lambda(\cdot) : \mathcal{X} \to \mathbb{R}_+$ defined by

$$\widehat\lambda(\boldsymbol{x}) \;=\; \sum_{t\in\mathcal{T}} \bar\lambda_t \; \mathbb{1}_{\{\boldsymbol{x}\in\mathcal{X}_t\}},$$

we also refer to (6.1) and (6.3). Note that this is not necessarily a strictly positive function. In a non strictly positive situation one may replace the MLE $\bar\lambda_t$ by the empirical credibility estimator $\bar\lambda_t^{\text{post}}$ given in (6.7) or by $\bar\lambda_t^{\text{B}}$ given in (6.8). We will discuss this just next.

Bayesian set-up

We have seen that there may be an issue with degenerate Poisson distributions on some of the leaves $t \in \mathcal{T}$. In applications one therefore replaces (6.11) by

$$(t, \varsigma_t) = \operatorname*{argmax}_{s\in\mathcal{T},\ \varsigma_s=(\mathcal{X}_{s0},\mathcal{X}_{s1})} \Delta\widetilde{D}^*_{\mathcal{X}_s}(\varsigma_s), \tag{6.13}$$

where

$$\Delta\widetilde{D}^*_{\mathcal{X}_t}(\varsigma_t) = D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar\lambda_t^{\text{B}}\right) - \left[D^*_{\mathcal{X}_{t0}}\left(\boldsymbol{N}, \bar\lambda_{t0}^{\text{B}}\right) + D^*_{\mathcal{X}_{t1}}\left(\boldsymbol{N}, \bar\lambda_{t1}^{\text{B}}\right)\right].$$

That is, we replace the MLEs $\bar\lambda_t$ by the empirical credibility estimators $\bar\lambda_t^{\text{B}}$ given in (6.8). This has the advantage that the resulting frequency estimators are always strictly positive, the disadvantage is that Lemma 6.4 does *not* hold true and errors may well be increasing by adding more splits. This is because only the MLE minimizes the corresponding Poisson deviance loss and maximizes the log-likelihood function, respectively. The SBS Tree Growing Algorithm with (6.11) replaced by (6.13) will, however, only consider SBS that lead to a real improvement (thanks to step (1b)) but it may miss the optimal one.

6.1.5 Example in motor insurance pricing, revisited

We revisit the synthetic MTPL data given in Appendix A and apply the SBS Tree Growing Algorithm of page 151 to estimate the expected frequency by (6.1).

Listing 6.1: R command `rpart`

```
1 tree <- rpart(claims ~ age + ac + power + gas + brand + area + dens + ct
2                        + offset(log(expo)),
3                        data=dat, method="poisson", parms=list(shrink=1),
4                        control=rpart.control(xval=10, minbucket=10000, cp=0.001))
5
6 rpart.plot(tree)
7 tree
```

We provide the R code in Listing 6.1. The variable `dat` contains the years at risk (`expo` $= v_i$), the `claims` counts N_i as well as all feature components. The method applied considers the Poisson deviance loss, and lines 3-4 of Listing 6.1 give the control variables, these are going to be described below. Due to the issue that we may receive MLE $\bar{\lambda}_{t\tau} = 0$ on some of the leaves, the algorithm `rpart` always uses the Bayesian approach. Note that `shrink` $= \gamma^{-1/2}$ is the shrinkage parameter used in the `rpart` command, see (6.9). The standard shrinkage parameter used is `shrink` $= 1$, which corresponds to $\gamma = 1$ in (6.9).

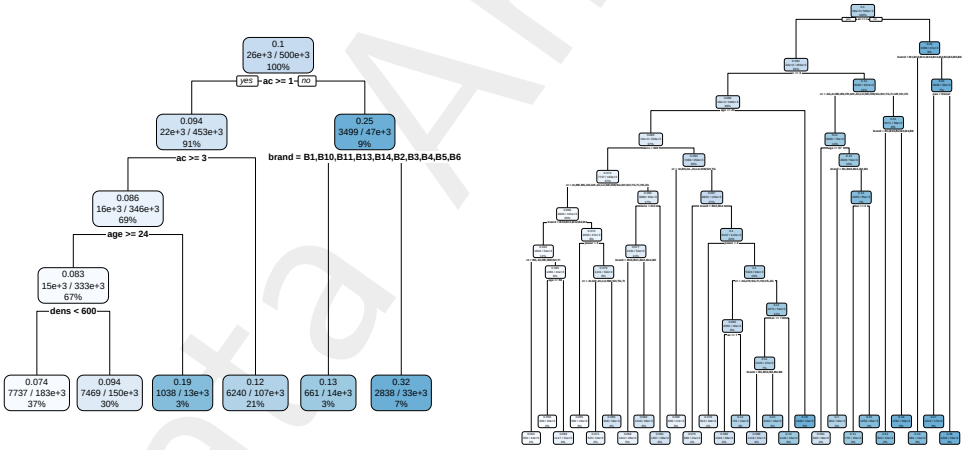


Figure 6.1: Regression trees for different stopping rules.

In Figure 6.1 we present the resulting regression trees of the SBS Tree Growing Algorithm using split criterion (6.13). The two trees (only) differ in their sizes because we have applied different stopping rules. We analyze the tree in Figure 6.1 (lhs), this tree is given in more detail in Listing 6.2.

We have used `n=500000` cases for its construction. Variable `node`) denotes the nodes of the tree, these indexes differ from our binary index in the sense that $t = 0$ corresponds to 1), $t = 00, 01$ correspond to 2) and 3), etc. `split` denotes the split criterion applied; `n` denotes the number of cases in that node; `deviance` is the resulting Poisson deviance loss

Listing 6.2: SBS Tree Growing Algorithm result

```
n= 500000

node), split, n, deviance, yval
      * denotes terminal node

 1) root 500000 145532.300 0.10269060
   2) ac>=0.5 452749 126855.500 0.09407058
     4) ac>=2.5 345628  93113.030 0.08592400
       8) age>=23.5 332748 87712.610 0.08277720
        16) dens< 599.5 183105 45721.540 0.07437700 *
        17) dens>=599.5 149643 41788.160 0.09374741 *
       9) age< 23.5 12880 4845.229 0.19376530 *
     5) ac< 2.5 107121 33147.840 0.12489500 *
   3) ac< 0.5 47251 16399.780 0.24966230
     6) brand=B1,B10,B11,B13,B14,B2,B3,B4,B5,B6 14444 3790.369 0.12806680 *
     7) brand=B12 32807 12073.280 0.32044710 *
```

of that node; `yval` gives the frequency estimate of that node; and `*` indicates whether the node is a leaf or not. For instance, line 9 in Listing 6.2 provides:

```
>    8) age >=23.5 332748 87712.610 0.08277720
```

This means that we consider node 8) that corresponds to $t = 0000$. This node was obtained from node $t = 000$ (equal to node 4)) using the split question `age>=23.5`. The node contains 332'748 cases, the Bayesian estimator is given by $\bar{\lambda}_t^{\mathrm{B}} = 0.08277720$ and the resulting Poisson deviance loss on that node $\mathcal{X}_t$ is $D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t^{\mathrm{B}}\right) = 87'712.610$. The graph in Figure 6.1 (lhs) gives the additional information that on node $t = 0000$ we have $N_t =$`15e+3` claims observed and that 67% of the cases fall into that node. Note that this information is not sufficient to calculate the full statistics because the volume w_t on node t is not displayed, see (6.5). The question that remains open for the moment is the stopping rule, i.e. should we rather use the small or the big tree in Figure 6.1? We treat this question in Section 6.2, below, together with an analysis of the predictive power of the regression tree estimator. This finishes the example for the moment.

6.1.6 Choice of categorical classes

In Section 2.4 the categorical classes for driver's age `age` and for the age of car `ac` were chosen based on pure expert opinion, see Figure 2.2 (lhs). We could also use a regression tree to determine these classes. We consider regression trees on the marginal observations.[2] This marginal consideration neglects possible interaction between the feature components but gives a data based answer on optimal (marginal) choices of (univariate) categorical classes for continuous feature components.
We choose for the two feature components exactly as many classes as in Figure 2.2 (lhs). The results are provided in Figure 6.2 and they give the classes shown in Table 6.1 (under the assumption that we would like to have 8 `age` classes and 4 `ac` distance classes). We

[2]Note that we use the same data for the construction of the categorical classes and the GLM analysis, of course, this might be questioned.

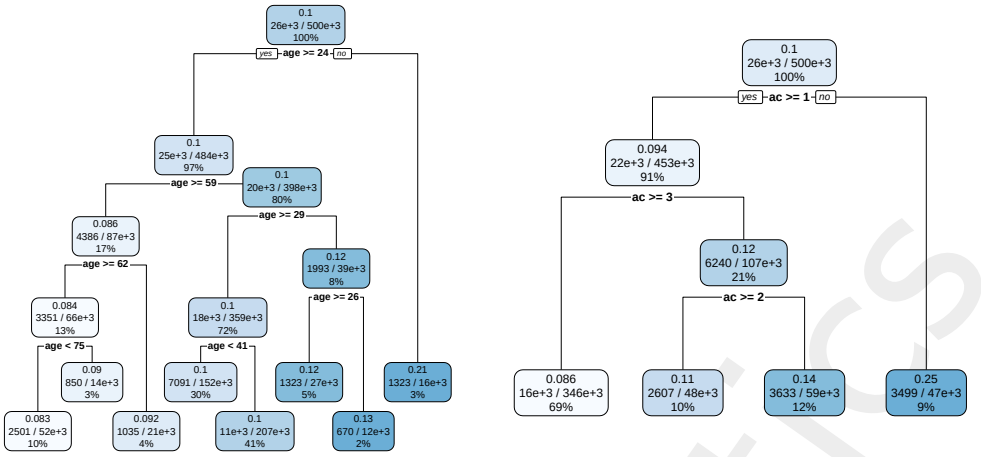


Figure 6.2: Choice of categorical classes using regression trees.

`age` class 1:	18-23
`age` class 2:	24-25
`age` class 3:	26-28
`age` class 4:	29-40
`age` class 5:	41-58
`age` class 6:	59-61
`age` class 7:	62-74
`age` class 8:	75-90

`ac` class 1:	0
`ac` class 2:	1
`ac` class 3:	2
`ac` class 4:	3+

Table 6.1: Optimal (lhs) `age` classes and (rhs) `ac` classes based on marginal regression tree estimators.

observe that the chosen `ac` classes are optimal, but for the `age` classes the regression tree algorithm proposes a slightly different choice of classes (which provides more marginal homogeneity in terms of Poisson deviance losses).

In Table 6.2 we consider the GLM analyses based on the two choices for the categorical `age` classes given in Figure 2.2 (lhs) and Table 6.1 (lhs), respectively. We observe a slight decrease in in-sample loss but at the same time an increase in the estimation loss. From this we conclude that in the present example we cannot gain much by a different/better choice of `age` classes.

Remark. This marginal choice of categorical classes can in particular be useful, if one starts from a huge set of categorical labels (e.g. grouping of industry sector codes) from which one would like to build bigger categorical classes. Doing this by expert opinion is often not feasible, but leaving the original categorical classes on a very granular level can be very time consuming in computational algorithms.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.3) GLM3	12.0s	50	28.2125	28.2133	0.4794	28.1937	10.2691%
(Ch6.0) GLM3.Tree	11.9s	50	–	–	0.4895	28.1905	10.2691%

Table 6.2: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 2.5.

6.2 Tree pruning

6.2.1 Binary trees and pruning

The resulting Poisson deviance loss of the partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of $\mathcal{X}$ is given by, see (6.12),

$$D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t\in\mathcal{T}}) \;=\; \sum_{t\in\mathcal{T}} D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right).$$

This describes an in-sample loss of the given tree estimate.[3] Corollary 6.5 states that any further SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$ reduces the Poisson deviance loss by $\Delta D^*_{\mathcal{X}_t}(\varsigma_t) \geq 0$. Since often a good stopping rule in the SBS Tree Growing Algorithm is not feasible, we present a different strategy of a tree selection here. The idea is to first grow a very large binary tree using the SBS Tree Growing Algorithm. In a second step this large binary tree is reduced by *pruning* the nodes of the large tree for which the reduction in Poisson deviance loss is not sufficient to justify that split.

Recall that T denotes the binary indexes of a binary tree with nodes $(\mathcal{X}_t)_{t\in\mathsf{T}}$. In order to not overload the language, we synonymously use T for the binary tree and/or its corresponding binary indexes here.

Choose a node with binary index $t \in \mathsf{T}$ being, say, $t = 0101$. This node belongs to generation $k = 3$, its *ancestors* (previous generations $k = 0, 1, 2$) are given by the binary indexes in

$$\{0, 01, 010\} \subset \mathsf{T},$$

and its *descendants* (later generations $k \geq 4$) are given by the binary indexes in

$$\left\{ s \in \mathsf{T};\; s = 0101\tau \;\text{ for some } \tau \in \bigcup_{\ell\in\mathbb{N}} \{0,1\}^\ell \right\}.$$

If we add to the latter set the binary index $t = 0101$ we obtain a new binary tree with t being its new root node (to be consistent in notation we could relabel the binary indexes, but we refrain from doing so, because we think that this is sufficiently clear from the

[3] To get in-sample loss $\mathcal{L}_D^{\text{is}}$ given in (1.10) we still need to scale with the number of observations n.

context). This motivates the following definition: choose $t \in \mathsf{T}$ and define the new binary tree indexes

$$\mathsf{T}_{(t+)} = \{t\} \cup \left\{ s \in \mathsf{T};\ s = t\tau \ \text{ for some } \tau \in \bigcup_{\ell \in \mathbb{N}} \{0,1\}^\ell \right\}. \tag{6.14}$$

These binary tree indexes $\mathsf{T}_{(t+)}$ define a new binary tree $(\mathcal{X}_s)_{s \in \mathsf{T}_{(t+)}}$ with root node t and with leaves indexed by $\mathcal{T}_{(t+)}$. The leaves provide a partition $(\mathcal{X}_s)_{s \in \mathcal{T}_{(t+)}}$ of the subset $\mathcal{X}_t$ of the total feature space $\mathcal{X}$.

Definition 6.6. *Pruning a binary tree at a node with binary index $t \in \mathsf{T}$ means that we delete all descendants of $\mathcal{X}_t$ from the tree and the pruned binary tree is obtained by the binary indexes*

$$\mathsf{T}_{(-t)} = \Big(\mathsf{T} \setminus \mathsf{T}_{(t+)}\Big) \cup \{t\} \ \subset\ \mathsf{T}.$$

The tree $\mathsf{T}_{(-t)}$ is the subtree of T that stops growing at binary index t, the tree $\mathsf{T}_{(t+)}$ is the subtree of T that has as root node the binary index t, and $\mathsf{T}_{(-t)} \cap \mathsf{T}_{(t+)} = \{t\}$. The general idea now is to start with a very large binary tree T. We then prune this large binary tree step by step by deleting all binary splits that do not substantially reduce the Poisson deviance loss (in relation to their complexity).

6.2.2 Minimal cost-complexity pruning

Theoretical results on smallest cost-optimal trees

An efficient way to obtain a pruning algorithm is to do minimal cost-complexity pruning which was invented by Breiman–Stone [17] and Breiman et al. [16].

Definition 6.7. *Choose a binary tree $(\mathcal{X}_t)_{t \in \mathsf{T}}$ with binary indexes T and resulting partition $(\mathcal{X}_t)_{t \in \mathcal{T}}$ of $\mathcal{X}$ indexed by $\mathcal{T}$. For $\eta \geq 0$, we define the cost-complexity measure of T by*

$$R_\eta(\mathsf{T}) = D^*(\boldsymbol{N}, (\bar\lambda_t)_{t \in \mathcal{T}}) + \eta |\mathcal{T}|.$$

Remarks 6.8.

- We aim at minimizing the Poisson deviance loss $D^*(\boldsymbol{N}, (\bar\lambda_t)_{t \in \mathcal{T}})$ by choosing a sufficiently large tree T, and in order to not choose an overly large tree we punish (regularize) the choices of large trees by a factor $\eta|\mathcal{T}|$ accounting for the number of leaves in that tree T. We call η the *regularization parameter*. Observe that this idea has already been used in GAMs, see (3.8), and we have been discussing ridge and LASSO regularization in Section 4.3.2. Here we replace the ridge (L^2) or the LASSO (L^1) regularization by the number of parameters (cardinality (L^0) regularization).

- In the above definition we use the Poisson deviance loss for the definition of the cost-complexity measure $R_\eta(\mathsf{T})$ of the tree T. In all what follows we could also use any other sensible objective function.

Every finite binary tree T has at most *finitely* many binary subtrees $\mathsf{T}' \subset \mathsf{T}$. Therefore, starting from a (very large) finite binary tree T, we can always find a binary subtree $\mathsf{T}'(\eta) \subset \mathsf{T}$ that minimizes the cost-complexity measure $R_\eta(\cdot)$ for a given regularization parameter $\eta \geq 0$. That is, there exists at least one minimizer $\mathsf{T}'(\eta)$ to the problem

$$\operatorname*{argmin}_{\mathsf{T}' \subset \mathsf{T}} R_\eta(\mathsf{T}'), \tag{6.15}$$

where the minimization $\mathsf{T}' \subset \mathsf{T}$ runs over all binary subtrees T' of the original tree T having identical root node $t = 0$. Optimization (6.15) can have more than one minimizer $\mathsf{T}'(\eta)$ for a given η. We will just prove that for every regularization parameter η there is a (unique) *smallest cost-optimal* binary subtree of T that solves (6.15). This smallest cost-optimal binary subtree of T will be denoted by $\mathsf{T}(\eta)$. The next theorem is proved in Section 6.4, below.

Theorem 6.9. *Choose a finite binary tree T and assume there is more than one minimizer $\mathsf{T}'(\eta)$ of (6.15) for a given $\eta \geq 0$. Then there exists a unique subtree $\mathsf{T}(\eta)$ of T that minimizes (6.15) and which satisfies $\mathsf{T}(\eta) \subset \mathsf{T}'(\eta)$ for all minimizers $\mathsf{T}'(\eta)$ of (6.15).*

We call this subtree $\mathsf{T}(\eta)$ the (unique) smallest cost-optimal binary subtree of T for the given regularization parameter $\eta \geq 0$.

Summarizing this result provides the following corollary.

Corollary 6.10. *The smallest (cost-optimal) minimizer $\mathsf{T}(\eta)$ of (6.15) is well-defined (and unique). We have for any minimizer $\mathsf{T}'(\eta)$ of (6.15) the relationships $\mathsf{T}(\eta) \subset \mathsf{T}'(\eta) \subset \mathsf{T}$ and*

$$R_\eta(\mathsf{T}(\eta)) = R_\eta(\mathsf{T}'(\eta)) = \min_{\mathsf{T}' \subset \mathsf{T}} R_\eta(\mathsf{T}').$$

Thus, we have for every regularization parameter $\eta \geq 0$ a smallest cost-optimal binary subtree $\mathsf{T}(\eta)$ of the original (large) binary tree T. The remaining question is: how can we determine these smallest cost-optimal subtrees efficiently? Searching the whole binary tree T for every η is too expensive.

Construction of the smallest cost-optimal subtree

Choose any binary subtree $\mathsf{T}' \subset \mathsf{T}$ with leaves $\mathcal{T}'$ and consider the function $r_{\mathsf{T}'} : \mathbb{R}_+ \to \mathbb{R}_+$ given by

$$\eta \;\mapsto\; r_{\mathsf{T}'}(\eta) = R_\eta(\mathsf{T}') = D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t \in \mathcal{T}'}) + \eta |\mathcal{T}'|. \tag{6.16}$$

Note that we introduce this new notation to indicate that $\eta \mapsto r_{\mathsf{T}'}(\eta)$ is a function of the regularization parameter η (for a given subtree T'), whereas $\mathsf{T} \mapsto R_\eta(\mathsf{T}')$ is a function of the subtrees T' (for a given regularization parameter η). The function $r_{\mathsf{T}'}(\eta)$ is linear in η with intercept $D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t \in \mathcal{T}'}) \geq 0$ and slope $|\mathcal{T}'| > 0$. We define the function

$$\eta \;\mapsto\; r(\eta) = \min_{\mathsf{T}' \subset \mathsf{T}} r_{\mathsf{T}'}(\eta) = \min_{\mathsf{T}' \subset \mathsf{T}} R_\eta(\mathsf{T}'), \tag{6.17}$$

which is well-defined because every finite binary tree T has at most finitely many binary subtrees T'.

Theorem 6.11. *Choose a finite binary tree* T. *The function* $r : \mathbb{R}_+ \to \mathbb{R}_+$ *defined in* (6.17) *is positive, strictly increasing, piece-wise linear and concave with finitely many values* $0 = \eta_0 < \eta_1 < \ldots < \eta_\kappa < \eta_{\kappa+1} = \infty$ *in which* $r(\cdot)$ *is not differentiable. Define* $\mathsf{T}^{(\ell)} = \mathsf{T}(\eta_\ell)$ *for* $\ell = 0, \ldots, \kappa$. *We have*

$$\mathsf{T} \supset \mathsf{T}^{(0)} \supset \ldots \supset \mathsf{T}^{(\kappa)} = \{0\}, \tag{6.18}$$

and for all $\ell = 0, \ldots, \kappa$

$$r(\eta) = r_{\mathsf{T}^{(\ell)}}(\eta) = R_\eta(\mathsf{T}^{(\ell)}) \qquad \text{for all } \eta \in [\eta_\ell, \eta_{\ell+1}).$$

This theorem is proved in Section 6.4, below, and illustrated in Figure 6.4 (lhs).

Theorem 6.11 gives the instructions for an efficient algorithm to find the smallest cost-optimal binary subtree $\mathsf{T}(\eta)$ of the original binary tree T for all regularization parameters $\eta \geq 0$. We just need to determine the tree sequence (6.18) for the selected η_ℓ's.

We start from a (large) binary tree T that has been generated by the SBS Tree Growing Algorithm on page 151. This algorithm has the property that it only uses SBS which lead to a real improvement in the Poisson deviance loss in (6.11), i.e. for the chosen SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ we have $\Delta D^*_{\mathcal{X}_t}(\varsigma_t) > 0$. For this reason we (may) initialize $\mathsf{T}^{(0)} = \mathsf{T}$, see also Remark 6.13 below. Assume that $\mathsf{T}^{(\ell)} \neq \{0\}$ has been constructed for the regularization parameter η_ℓ; if $\mathsf{T}^{(\ell)}$ is the root tree we have found the final $\ell = \kappa$ and stop the algorithm. To construct $\mathsf{T}^{(\ell+1)} \subset \mathsf{T}^{(\ell)}$ and determine the corresponding $\eta_{\ell+1} > \eta_\ell$ we consider for any $t \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ the (root) tree $\{t\}$ and the (non-trivial) binary subtree $\mathsf{T}^{(\ell)}_{(t+)} \subset \mathsf{T}^{(\ell)}$. Since every SBS considered is assumed to lead to a real improvement in the Poisson deviance loss we have

$$D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\{t\}}) - D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(t+)}}) > 0.$$

As long as[4]

$$r_{\{t\}}(\eta) = R_\eta(\{t\}) \;>\; R_\eta(\mathsf{T}^{(\ell)}_{(t+)}) = r_{\mathsf{T}^{(\ell)}_{(t+)}}(\eta), \tag{6.19}$$

the root tree $\{t\}$ has a higher cost-complexity than $\mathsf{T}^{(\ell)}_{(t+)}$. Therefore we do not stop growing the tree in node t. But as soon as the last inequality becomes an equality (by increasing the regularization parameter η) we should prune the tree $\mathsf{T}^{(\ell)}_{(t+)}$ to the simpler root tree $\{t\}$ for that increased η. The idea now is to find the *weakest node* $t^*_{\ell+1} \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ defined by

$$t^*_{\ell+1} = \operatorname*{argmin}_{t\in\mathsf{T}^{(\ell)}\setminus\mathcal{T}^{(\ell)}} \frac{D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\{t\}}) - D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(t+)}})}{|\mathcal{T}^{(\ell)}_{(t+)}| - 1}. \tag{6.20}$$

This weakest node determines the minimal regularization parameter over all nodes in $\mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ to turn inequality (6.19) into an equality for exactly this node and this regularization parameter. We prune the tree $\mathsf{T}^{(\ell)}$ at this weakest node $t^*_{\ell+1}$. The next lemma is proved in Section 6.4, below.

[4]Note that we use an abuse of notation in (6.19) because we interpret the functions on trees with root node $\{t\}$; for more clarification we also refer to (6.37), below.

Lemma 6.12. *Choose $\ell \geq 0$ with $\mathsf{T}^{(\ell)} \neq \{0\}$. The weakest node $t^*_{\ell+1}$ defined by (6.20) provides $\mathsf{T}^{(\ell+1)} = \mathsf{T}^{(\ell)}_{(-t^*_{\ell+1})}$ and*

$$\begin{aligned}
\eta_{\ell+1} &= \frac{D^*_{\mathcal{X}_{t^*_{\ell+1}}}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\{t^*_{\ell+1}\}}) - D^*_{\mathcal{X}_{t^*_{\ell+1}}}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(t^*_{\ell+1}+)}})}{|\mathcal{T}^{(\ell)}_{(t^*_{\ell+1}+)}| - 1} \\
&= \frac{D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(-t^*_{\ell+1})}}) - D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}})}{|\mathcal{T}^{(\ell)}| - |\mathcal{T}^{(\ell)}_{(-t^*_{\ell+1})}|} \\
&= \frac{D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell+1)}}) - D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}})}{|\mathcal{T}^{(\ell)}| - |\mathcal{T}^{(\ell+1)}|}.
\end{aligned}$$

If $\mathsf{T}^{(\ell+1)} = \{0\}$ we set $\kappa = \ell + 1$.

Remark. In fact, Lemma 6.12 uses an induction in its proof, which shows that the algorithm is well initialized for $\ell = 0$ and it also determines the maximal constant κ.

Minimal Cost-Complexity Pruning Algorithm.

---

(0) Initialization: Set $\mathsf{T}^{(0)} = \mathsf{T}$.

(1) Repeat until $\mathsf{T}^{(\ell)}$ is the root tree $\{0\}$:

  (a) find the weakest node $t^*_{\ell+1} \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ and define $\mathsf{T}^{(\ell+1)} \subset \mathsf{T}^{(\ell)}$ and $\eta_{\ell+1} > \eta_\ell$ as in Lemma 6.12;

  (b) if $\mathsf{T}^{(\ell+1)} = \{0\}$ set $\kappa = \ell + 1$.

(2) Return $(\eta_\ell)_{\ell=0,\dots,\kappa}$ and $(\mathsf{T}^{(\ell)})_{\ell=0,\dots,\kappa}$.

Remark 6.13. The initialization step (0) sets $\mathsf{T}^{(0)} = \mathsf{T}$. Since we grow the tree T by the SBS Tree Growing Algorithm, only SBS are considered that strictly improve the Poisson deviance loss. From this it immediately follows that the smallest cost-optimal tree for $\eta = \eta_0 = 0$ is given by the original tree T. In a situation where this is not the case, i.e. where another tree growing algorithm has been used that does not have this property, one needs to already prune the starting tree, see Breiman et al. [16].

Regularization and cost-complexity parameters

The parameter $\eta \geq 0$ regularizes the complexity of the resulting smallest cost-optimal binary subtree $\mathsf{T}(\eta)$, see Definition 6.7. For an arbitrary binary tree T the cost-complexity measure in η is given by

$$R_\eta(\mathsf{T}) = D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\mathcal{T}}) + \eta|\mathcal{T}|.$$

From this we see that the regularization parameter η has the same unit as the Poisson deviance loss. Since this cost-complexity consideration can be applied to any loss function, one often normalizes the regularization by the corresponding loss of the root tree, i.e. one considers

$$\begin{aligned} R_\eta(\mathsf{T}) &= D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\mathcal{T}}) + \eta|\mathcal{T}| \\ &= D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\mathcal{T}}) + \text{cp } D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\{0\}})\ |\mathcal{T}|, \end{aligned}$$

where $D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\{0\}})$ is the Poisson deviance loss of the root tree with (homogeneous) MLE $\bar\lambda_0 = \sum_{i=1}^n N_i / \sum_{i=1}^n v_i$, and we define the *cost-complexity parameter*

$$\text{cp} = \frac{\eta}{D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\{0\}})}.$$

The sequence $\eta_{\ell+1}$, $\ell = 0, \ldots, \kappa-1$, given by

$$\eta_{\ell+1} = \frac{D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell+1)}}) - D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}})}{|\mathcal{T}^{(\ell)}| - |\mathcal{T}^{(\ell+1)}|},$$

then provides cost-complexity parameters

$$\begin{aligned} \text{cp}_{\ell+1} &= \frac{\eta_{\ell+1}}{D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\{0\}})} \\ &= \frac{1}{D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\{0\}})} \frac{D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell+1)}}) - D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}})}{|\mathcal{T}^{(\ell)}| - |\mathcal{T}^{(\ell+1)}|}. \end{aligned} \tag{6.21}$$

Remark. Note that we call η regularization parameter and its normalized counterpart cp cost-complexity parameter.

Example in motor insurance pricing, revisited

We revisit the example of Section 6.1.5, but for illustrative purposes we start with a small tree. This small tree T is given in Figure 6.3 and Listing 6.3, note that this small tree is a subtree of the trees in Figure 6.1.

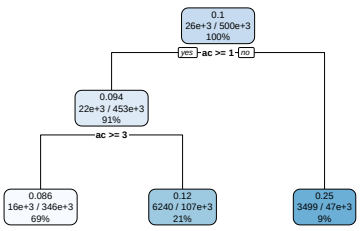


Figure 6.3: Regression tree T for an early stopping rule.

This tree T has leaf indexes $\{000, 001, 01\}$, and it has two subtrees with root 0 and with leaf indexes $\{00, 01\}$ and $\{0\}$, the latter is the root tree. We denote these trees for the moment by $\mathsf{T}_3, \mathsf{T}_2, \mathsf{T}_1$ (the first one denoting the full tree T and the last one denoting

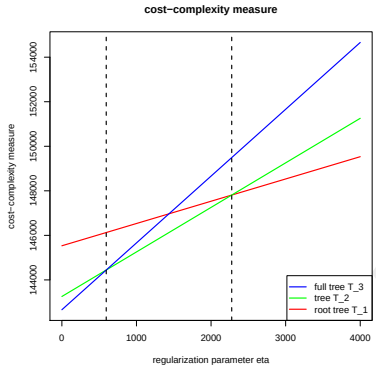
Listing 6.3: Regression tree T for an early stopping rule

```
1  n= 500000
2
3  node), split, n, deviance, yval
4        * denotes terminal node
5
6  1) root 500000 145532.30 0.10269060
7    2) ac>=0.5 452749 126855.50 0.09407058
8      4) ac>=2.5 345628  93113.03 0.08592400 *
9      5) ac< 2.5 107121  33147.84 0.12489500 *
10   3) ac< 0.5 47251  16399.78 0.24966230 *
```

the root tree). For these three trees T_i, $i = 1, 2, 3$, we can study the cost-complexity measures

$$\eta \;\mapsto\; r_{\mathsf{T}_i}(\eta) = R_\eta(\mathsf{T}_i) = D^*(\boldsymbol{N}, (\bar\lambda_t)_{t\in\mathcal{T}_i}) + \eta|\mathcal{T}_i|, \tag{6.22}$$

where $\mathcal{T}_i$ are the leaves of T_i with $|\mathcal{T}_i| = i$ for $i = 1, 2, 3$.



cp	nsplit	rel error	xerror	xstd
0.0156463	0	1.00000	1.00001	0.0044701
0.0040858	1	0.98435	0.98437	0.0043582
0.0039000	2	0.98027	0.98139	0.0043392

Figure 6.4: (lhs) Cost-complexity measures (6.22) for the three trees $\mathsf{T}_3, \mathsf{T}_2, \mathsf{T}_1$ and (rhs) resulting cost-complexity results.

In Figure 6.4 (lhs) we plot the cost-complexity measures (6.22) for our example given in Figure 6.3. We see the increasing, piece-wise linear and concave property of $r(\cdot)$, see Theorem 6.11, and that

$$\eta_1 = 595, \qquad \eta_2 = \eta_\kappa = 2'277, \qquad \mathrm{cp}_1 = 0.0040858, \qquad \mathrm{cp}_2 = \mathrm{cp}_\kappa = 0.0156463.$$

These values are obtained from the R command `printcp(tree)` where `tree` is obtained as in Listing 6.1. Note that on line 4 of Listing 6.1 we specify the parameter `cp` which determines to which cost-complexity parameter we grow the tree. If we set `cp=0.0039` we exactly receive Figure 6.4 (rhs). The first column specifies the critical cost-complexity parameters cp_1 and cp_2, (6.21) then provides the corresponding regularization parameters η_1 and η_2. Decreasing `cp` in Listing 6.1 will provide a larger tree.
Moreover, we only consider splits so that each leaf contains at least `minbucket=10000` cases, see line 4 of Listing 6.1. We choose this value comparably large because if

$$v = \sum_{i=1}^{10'000} v_i = 5000 \qquad \text{and} \qquad \widehat\lambda = 10\%,$$

then we have estimated confidence bounds of two standard deviations given by, see also (2.20),

$$\left[\widehat{\lambda} - 2\,\sqrt{\widehat{\lambda}/v},\ \widehat{\lambda} + 2\,\sqrt{\widehat{\lambda}/v}\right] \ = \ [9.1\%, 10.9\%]\,.$$

That is, we obtain a precision of roughly 0.9% in the expected frequency estimate $\widehat{\lambda}$ (which is not very precise). The column `rel error` gives a scaled version of the χ^2-test statistics, see (2.15), given by

$$\texttt{rel error}_i = \frac{D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t\in\mathcal{T}_i})}{D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t\in\{0\}})},$$

for $i = 1, 2, 3$. Note that this relative error is an in-sample loss. We observe that the in-sample loss decays by roughly 2% by the first two splits.

The remaining question is the best choice of the regularization parameter $\eta \geq 0$. This will be done in Section 6.2.4, below.

6.2.3 Choice of the best pruned tree

The remaining difficulty is to find the optimal regularization parameter η_ℓ, $\ell = 0, \ldots, \kappa$, for pruning, i.e. which one of the optimally pruned (smallest cost-optimal) trees $\mathsf{T} = \mathsf{T}^{(0)} \supset \mathsf{T}^{(1)} \supset \ldots \supset \mathsf{T}^{(\kappa-1)} \supset \mathsf{T}^{(\kappa)} = \{0\}$ should be selected. The choice of the best pruned tree is often done by K-fold cross-validation. Assume that we use (stratified) K-fold cross-validation for the Poisson deviance loss, see (1.11),

$$\mathcal{L}_D^{\mathrm{CV}}(\eta) = \frac{1}{K}\ \sum_{k=1}^{K} \frac{1}{|\mathcal{B}_k|} \sum_{i\in\mathcal{B}_k} 2\ N_i \left[\frac{\widehat{\lambda}^{(-\mathcal{B}_k)}(\boldsymbol{x}_i)v_i}{N_i} - 1 - \log\left(\frac{\widehat{\lambda}^{(-\mathcal{B}_k)}(\boldsymbol{x}_i)v_i}{N_i}\right)\right], \tag{6.23}$$

where $\widehat{\lambda}^{(-\mathcal{B}_k)}(\cdot) = \widehat{\lambda}^{(-\mathcal{B}_k)}(\ \cdot\ ; \eta)$ is the regression tree estimator

$$\widehat{\lambda}^{(-\mathcal{B}_k)}(\boldsymbol{x}) = \widehat{\lambda}^{(-\mathcal{B}_k)}(\boldsymbol{x}; \eta) \ = \sum_{t\in\mathcal{T}^{(-\mathcal{B}_k)}(\eta)} \bar{\lambda}_t\ \mathbb{1}_{\{\boldsymbol{x}\in\mathcal{X}_t\}},$$

on the optimally pruned tree $\mathsf{T}^{(-\mathcal{B}_k)}(\eta) \subset \mathsf{T}^{(-\mathcal{B}_k)}$ for regularization parameter η, where for the construction of the (big) tree $\mathsf{T}^{(-\mathcal{B}_k)}$ we have only used the (stratified) training data $\mathcal{D}^{(-\mathcal{B}_k)} = \mathcal{D} \setminus \mathcal{D}^{\mathcal{B}_k}$, and data $\mathcal{D}^{(\mathcal{B}_k)}$ is used for validation. This optimally pruned tree provides an η- and $\mathcal{D}^{(-\mathcal{B}_k)}$-dependent partition $(\mathcal{X}_t)_{t\in\mathcal{T}^{(-\mathcal{B}_k)}(\eta)}$ of the feature space $\mathcal{X}$ and the corresponding estimators $(\bar{\lambda}_t)_{t\in\mathcal{T}^{(-\mathcal{B}_k)}(\eta)}$ on that partition.

This (stratified) K-fold cross-validation error $\mathcal{L}_D^{\mathrm{CV}}(\eta)$ can be considered as an out-of-sample prediction error estimate (using the Poisson deviance loss as objective function) on a portfolio consisting of the characteristics given by data $\mathcal{D}$ and using an optimally pruned binary tree for the regularization parameter η. That is, we estimate the out-of-sample loss of the optimally pruned tree $\mathsf{T}(\eta)$ at regularization parameter η by

$$\widehat{\mathcal{L}}_D^{\mathrm{oos}}(\eta) = \mathcal{L}_D^{\mathrm{CV}}(\eta). \tag{6.24}$$

Note that we deliberately use the hat-notation on the left-hand side: the smallest cost-optimal tree $\mathsf{T}(\eta)$ and the original tree T were constructed on the entire $\mathcal{D}$. In order to estimate the resulting out-of-sample loss we use (stratified) K-fold cross-validation. This cross-validation separates the data into training samples $\mathcal{D}^{(-\mathcal{B}_k)}$ and validation samples $\mathcal{D}^{\mathcal{B}_k}$, $k = 1, \ldots, K$, from which the cross-validation error of the corresponding smallest cost-optimal trees $\mathsf{T}^{(-\mathcal{B}_k)}(\eta)$ is calculated. That is, the right-hand and the left-hand sides of (6.24) do not use the data $\mathcal{D}$ in exactly the same manner because cross-validation requires a training and a validation sample for back-testing (estimating to out-of-sample loss on a test data set). Therefore, we consider the right-hand side as an estimate of the left-hand side.

Choice of the Best Pruned Tree.

---

(0) Grow a large tree T on the entire data $\mathcal{D}$ using the SBS Tree Growing Algorithm.

(1) Determine the regularization parameters $\eta_0, \ldots, \eta_{\kappa+1}$, see Lemma 6.12, and define their geometric means for $\ell = 0, \ldots, \kappa$ by

$$\check{\eta}_\ell = \sqrt{\eta_\ell \eta_{\ell+1}}, \qquad \text{with } \check{\eta}_\kappa = \infty.$$

(2) Partition the data $\mathcal{D}$ into (stratified) subsets $\mathcal{D}^{\mathcal{B}_k}$, $k = 1, \ldots, K$, for (stratified) K-fold cross-validation:

   (a) grow for every k a large tree $\mathsf{T}^{(-\mathcal{B}_k)}$ based on the (stratified) training sample $\mathcal{D} \setminus \mathcal{D}^{\mathcal{B}_k}$;

   (b) determine for every k the optimally pruned subtrees $\mathsf{T}^{(-\mathcal{B}_k)}(\check{\eta}_\ell)$, $\ell = 0, \ldots, \kappa$;

   (c) calculate the cross-validation error estimate $\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell)$ for every $\ell = 0, \ldots, \kappa$ based on the optimally pruned subtrees $\mathsf{T}^{(-\mathcal{B}_k)}(\check{\eta}_\ell)$ and the corresponding validation samples $\mathcal{D}^{\mathcal{B}_k}$, $k = 1, \ldots, K$.

(3) Determine $\ell^* \in \{0, \ldots, \kappa\}$ by one of the following criteria:

   – *minimum cross-validation error*: choose ℓ^* such that $\check{\eta}_{\ell^*}$ minimizes (6.24) for $\ell = 0, \ldots, \kappa$; we denote this minimizing index by $\ell_{\min} = \ell^*$;

   – *1-SD rule*: the 1 standard deviation rule acknowledges that there is some uncertainty involved in the estimate (6.23)-(6.24); we therefore provide an estimate for the standard deviation $\widehat{\mathrm{Var}}(\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell))^{1/2}$ (which is done empirically over the K cross-validation samples); the 1-SD rule proposes to choose ℓ^* minimal such that

$$\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_{\ell^*}) \;\leq\; \min_\ell \mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell) + \widehat{\mathrm{Var}}(\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell))^{1/2};$$

   – *elbow criterion*: when looking at the sequence $\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell)$, $\ell = \kappa, \ldots, 0$, one often observes that it steeply drops in ℓ, then has a kink at a certain index ℓ_E and after that kink it is rather flat; the elbow criteria suggests to choose $\ell^* = \ell_E$.

(4) Return the best pruned tree $\mathsf{T}(\eta_{\ell^*})$ (for one of the previous criteria).

Remark.

- The geometric means $\check{\eta}_\ell = \sqrt{\eta_\ell \eta_{\ell+1}}$ are considered to be typical values on the intervals $[\eta_\ell, \eta_{\ell+1})$. Note that the optimal regularization parameters of Lemma 6.12 will vary over the choices $\mathsf{T}^{(-\mathcal{B}_k)}$, $k = 1, \ldots, K$. Therefore, 'typical' choices $\check{\eta}_\ell$, $\ell = 0, \ldots, \kappa$, based on all data $\mathcal{D}$ are considered for cross-validation, i.e. around $\check{\eta}_\ell$ we expect the critical tree size of the optimal tree to be more stable because these values are sufficiently different from η_ℓ and $\eta_{\ell+1}$ where the tree sizes change.
- There is still the flexibility of the choice of the selection criteria in step (3). This is further discussed in the examples.
- There is still the freedom of growing the (large) trees T and $\mathsf{T}^{(-\mathcal{B}_k)}$, $k = 1, \ldots, K$. If we use the SBS Tree Growing Algorithm we typically use the same criterion for the determination of the algorithm for all these trees.

6.2.4 Example in motor insurance pricing, revisited

We consider two examples. The first one will be based on the small tree T of Figure 6.3. For the second example we will grow a large tree that we are going to prune.

Small tree of Figure 6.3

We continue the example of Figure 6.3: for illustrative purposes it is based on the small tree $\mathsf{T} = \mathsf{T}_3$, only. For this small tree we have, see also Figure 6.4,

$$\check{\eta}_0 = 0, \qquad \check{\eta}_1 = 1'164 \qquad \text{and} \qquad \check{\eta}_2 = \infty.$$

The column `rel error` in Figure 6.4 (rhs) gives an in-sample loss that is generally too small because of potential over-fitting to the learning data. The column `xerror` in Figure 6.4 (rhs) then provides the relative 10-fold cross-validation errors (note that we choose `xval` $= K = 10$ in Listing 6.1) for these regularization parameters $\check{\eta}_\ell$, that is,

$$\texttt{xerror}_\ell = \frac{n \mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell)}{D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t \in \{0\}})},$$

for $\ell = 1, 2, 3$. According to the minimum cross-validation error rule we would choose the maximal tree T_3 in this example. The final column `xstd` in Figure 6.4 (rhs) denotes the relative estimated standard error of the cross-validation, i.e.

$$\texttt{xstd}_\ell = \frac{n \widehat{\mathrm{Var}}(\mathcal{L}_D^{\mathrm{CV}}(\check{\eta}_\ell))^{1/2}}{D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t \in \{0\}})},$$

for $\ell = 1, 2, 3$. Note that these numbers are displayed in gray color in Figure 6.4 (rhs) because they are not calculated by version 4.1.10 of `rpart` as we require.[5] Therefore,

cp	nsplit	rel error	xerror	xstd not stratified	xstd stratified	strat. CV $\mathcal{L}_D^{\text{is}}$
0.0156463	0	1.00000	1.00001	0.01076	0.00218	29.1065
0.0040858	1	0.98435	0.98437	0.01120	0.00397	28.6516
0.0039000	2	0.98027	0.98139	0.00992	0.00402	28.5649

Table 6.3: Cost-complexity and 10-fold cross-validation outputs for the trees $\mathsf{T}_3, \mathsf{T}_2, \mathsf{T}_1$, see Figure 6.3, the last column is in 10^{-2}.

we have implemented our own cross-validation procedure. The results are given in Table 6.3. We observe a substantial decrease in `xstd` when applying stratified cross-validation compared to non-stratified one, thus, uncertainty in cross-validation errors substantially decreases when applying the stratified version. This observation is fully in line with Figure 5.8 (middle, rhs). The final column in Table 6.3 displays the (absolute) 10-fold cross-validation errors (in 10^{-2}). These should be compared to Table 5.4 which provides the values for the models GLM4, GAM3 and DNN1-3. We observe that the tree T_3 cannot compete with the models of Table 5.4, nevertheless the first two splits lead to a remarkable decrease in Poisson deviance loss. That this small tree is not competitive is not surprising because we did not grow a sufficiently large tree (and applied optimal pruning). This we are going to do next.

Optimally pruned large tree

Now we grow a large tree. We therefore use the R command given in Listing 6.1, but we control the size of the tree by setting `cp=0.000001` and `minbucket=1000`, i.e. we grow a large three $\mathsf{T} = \mathsf{T}^{(0)}$ having in each leaf at least 1'000 cases.
The stratified 10-fold cross-validation results are given in Figure 6.5. We see a decrease in cross-validation losses for the first 70 SBS (green vertical line in Figure 6.5). From the 71st split on the cross-validation losses start to increase which indicates over-fitting to the data. The error bars correspond to 1 standard deviation obtained by stratified 10-fold cross-validation. Using the 1-SD rule we obtain a comparably small tree with 13 leaves (blue line in Figure 6.5), using the minimum cross-validation error we choose a tree with 71 leaves (green line in Figure 6.5). These trees are shown in Figure 6.6. We call these binary trees $\mathsf{T}^{\text{1-SD}}$ and $\mathsf{T}^{\min}$ for further reference.
For model comparison we then compare the resulting prediction errors, these are provided in Table 6.4. We observe that the 1-SD rule regression tree results provide higher estimation errors than the models GLM4 and GAM3 which shows that this regression tree $\mathsf{T}^{\text{1-SD}}$ is not competitive. In fact, this is often observed that the 1-SD rule tends to go for a tree that is too small.
The minimum cross-validation error tree $\mathsf{T}^{\min}$ performs much better. It has a better performance than model GAM3 in terms of estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$. This illustrates (once more) that the GAM is missing important (non-multiplicative) interactions in the regression function. On the other hand, this regression tree has a worse performance than the neural network approaches. This is also a common observation. The neural network

[5] In our implementation we would like to use the empirical frequencies for choosing stratified samples.

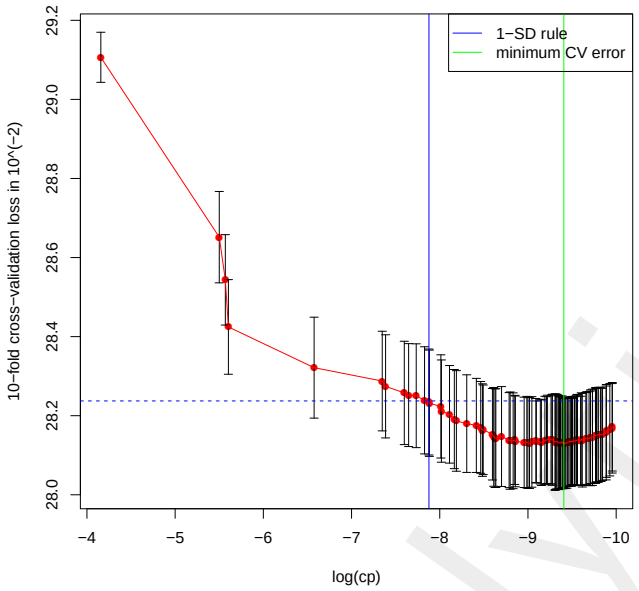


Figure 6.5: Cross-validation results from growing a large binary tree, error bars correspond to 1 standard deviation obtained by stratified 10-fold cross-validation.

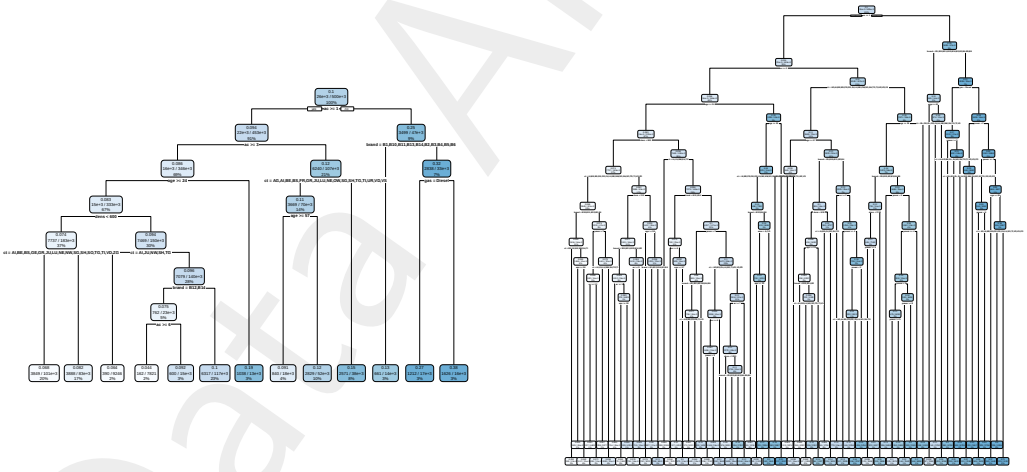


Figure 6.6: Optimally pruned trees according to the 1-SD rule $\mathsf{T}^{\text{1-SD}}$ and the minimum cross-validation error rule $\mathsf{T}^{\min}$, see also Figure 6.5.

architecture with differentiable activation function can continuously inter- and extrapolate for continuous feature values. This makes it more powerful than the regression tree approach because the neural network is less sensitive in small (homogeneous) subportfolios as long as it can benefit from an ordinal relationship within feature components, i.e. can "continuously learn over neighboring leaves".

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.4) CANN1	28s	703†	27.9292	27.9362	0.2284	27.8940	10.1577%
(Ch5.5) CANN2	27s	780†	27.9306	27.9456	0.2092	27.8684	10.2283%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%
(Ch5.7) blended DNN	–	–	–	–	0.1336	27.6939	10.2691%
(Ch6.1) Tree1 $\mathsf{T}^{\text{1-SD}}$	65s‡	13	–	28.2280	0.4814	28.1698	10.2674%
(Ch6.2) Tree2 $\mathsf{T}^{\min}$	65s‡	71	–	28.1388	0.3748	27.9156	10.2570%

Table 6.4: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration (‡ includes 10-fold cross-validation to select the optimal tree), and '# param.' gives the number of estimated model parameters († only considers the network parameters and not the non-trainable GAM parameters), this table follows up from Table 5.8.

6.3 Binary tree classification

Above we have introduced the SBS tree construction for the Poisson regression problem which provided us regression tree estimator (6.1). We now turn to the classification problem. Assume that the cases $(Y, \boldsymbol{x})$ have a categorical distribution satisfying

$$\pi_y(\boldsymbol{x}) = \mathbb{P}[Y = y] \geq 0 \qquad \text{for } y \in \mathcal{Y} = \{0, \ldots, J-1\},$$

with normalization $\sum_{y \in \mathcal{Y}} \pi_y(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \mathcal{X}$. The classifier $\mathcal{C} : \mathcal{X} \to \mathcal{Y}$ provides for cases $(Y, \boldsymbol{x})$ the labels

$$\mathcal{C}(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ \pi_y(\boldsymbol{x}), \tag{6.25}$$

with a deterministic rule if the maximum is not unique. This is similar to the logistic regression classification in Section 2.5.2. However, in this section we do not assume any structural form for the probabilities $\pi_y(\boldsymbol{x})$ in terms of the features $\boldsymbol{x}$, but we aim at estimating and approximating this structure with a SBS tree construction.

6.3.1 Empirical probabilities

Assume we have data $\mathcal{D}$ of the form (2.22). Choose a subset $\mathcal{X}' \subset \mathcal{X}$ of the feature space. Denote by $n(y, \mathcal{X}'; \mathcal{D})$ the number of cases $(Y_i, \boldsymbol{x}_i)$ in $\mathcal{D}$ that have feature $\boldsymbol{x}_i \in \mathcal{X}'$ and response $Y_i = y$. The (empirical) probability that a randomly chosen case $(Y, \boldsymbol{x})$ of $\mathcal{D}$ belongs to feature set $\mathcal{X}'$ and has response $Y = y$ is given by

$$\widehat{p}(y, \mathcal{X}') = \widehat{p}(y, \mathcal{X}'; \mathcal{D}) = \frac{n(y, \mathcal{X}'; \mathcal{D})}{\sum_{y' \in \mathcal{Y}} n(y', \mathcal{X}; \mathcal{D})} = \frac{n(y, \mathcal{X}'; \mathcal{D})}{n}. \tag{6.26}$$

We use the hat-notation for the empirical probabilities $\widehat{p}$ to indicate that these depend on the data $\mathcal{D}$. The (empirical) probability that a randomly chosen case $(Y, \boldsymbol{x})$ belongs to $\mathcal{X}'$ is given by

$$\widehat{p}(\mathcal{X}') = \sum_{y \in \mathcal{Y}} \widehat{p}(y, \mathcal{X}') = \frac{\sum_{y \in \mathcal{Y}} n(y, \mathcal{X}'; \mathcal{D})}{\sum_{y \in \mathcal{Y}} n(y, \mathcal{X}; \mathcal{D})} = \frac{\sum_{y \in \mathcal{Y}} n(y, \mathcal{X}'; \mathcal{D})}{n},$$

and the (empirical) probability that a randomly chosen case $(Y, \boldsymbol{x})$ in $\mathcal{X}'$ has response y is given by

$$\widehat{p}(y|\mathcal{X}') = \frac{\widehat{p}(y, \mathcal{X}')}{\widehat{p}(\mathcal{X}')} = \frac{n(y, \mathcal{X}'; \mathcal{D})}{\sum_{y' \in \mathcal{Y}} n(y', \mathcal{X}'; \mathcal{D})}, \tag{6.27}$$

supposed that $\widehat{p}(\mathcal{X}') > 0$, i.e. there are cases in $\mathcal{X}'$.

Remark. These empirical probabilities $\widehat{p}$ purely depend on the data $\mathcal{D}$. The theory also works in the setting where one has prior knowledge about the classification probabilities and, therefore, uses a modified version for the definition of $\widehat{p}$. We will meet the modified framework in Chapter 7 for the AdaBoost algorithm, for more details we also refer to Breiman et al. [16].

6.3.2 Standardized binary split tree growing algorithm for classification

Similar to Section 6.1.3 we need a goodness of split criterion which evaluates possible SBS of the feature space. The criterion that we consider first relates to misclassification.

Class assignment rule

We start by defining the class assignment rule. Assume we have a partition $(\mathcal{X}_t)_{t \in \mathcal{T}}$ of the feature space $\mathcal{X}$ (with every $\mathcal{X}_t$ containing at least one case of the data $\mathcal{D}$). We define the class assignment rule

$$y_t^* = \operatorname*{argmax}_{y^* \in \mathcal{Y}} \widehat{p}(y^*|\mathcal{X}_t) = \operatorname*{argmax}_{y^* \in \mathcal{Y}} n(y^*, \mathcal{X}_t; \mathcal{D}). \tag{6.28}$$

That is, we choose the label y_t^* on $\mathcal{X}_t$ that has the biggest likelihood (with a deterministic rule if there is more than one). This provides us classifier estimator $\widehat{\mathcal{C}} : \mathcal{X} \to \mathcal{Y}$

$$\widehat{\mathcal{C}}(\boldsymbol{x}) \;=\; \sum_{t \in \mathcal{T}} y_t^* \; \mathbb{1}_{\{\boldsymbol{x} \in \mathcal{X}_t\}}. \tag{6.29}$$

This assigns a class $\widehat{\mathcal{C}}(\boldsymbol{x}) \in \mathcal{Y}$ to every feature $\boldsymbol{x} \in \mathcal{X}$.

Misclassification rate

Next we introduce the misclassification rate of class assignment rule (6.29) by

$$\mathcal{L}^{\text{is}}_{\mathbb{1}_{\{\neq\}}}(\mathcal{D}, (y_t^*)_{t \in \mathcal{T}}) = \frac{1}{n} \sum_{i=1}^{n} \; \mathbb{1}_{\{Y_i \neq \widehat{\mathcal{C}}(\boldsymbol{x}_i)\}}. \tag{6.30}$$

Note that this is in line with (2.31) and determines the in-sample misclassification rate. We can represent this misclassification rate as given in the following lemma.

Lemma 6.14. *The misclassification rate satisfies*

$$\mathcal{L}^{\text{is}}_{\mathbb{1}_{\{\neq\}}}(\mathcal{D},(y_t^*)_{t\in\mathcal{T}}) = \sum_{t\in\mathcal{T}} \widehat{p}(\mathcal{X}_t)\,(1-\widehat{p}(y_t^*|\mathcal{X}_t))\,.$$

Proof. We can exchange the summation in the definition of the misclassification rate which provides

$$\begin{aligned}
\mathcal{L}^{\text{is}}_{\mathbb{1}_{\{\neq\}}}(\mathcal{D},(y_t^*)_{t\in\mathcal{T}}) &= \frac{1}{n}\sum_{i=1}^n \mathbb{1}_{\{Y_i\neq\widehat{\mathcal{C}}(\boldsymbol{x}_i)\}} = \frac{1}{n}\sum_{i=1}^n \sum_{t\in\mathcal{T}} \mathbb{1}_{\{\boldsymbol{x}_i\in\mathcal{X}_t\}}\mathbb{1}_{\{Y_i\neq y_t^*\}} \\
&= \frac{1}{n}\sum_{t\in\mathcal{T}}\sum_{i=1}^n \mathbb{1}_{\{\boldsymbol{x}_i\in\mathcal{X}_t\}}\mathbb{1}_{\{Y_i\neq y_t^*\}} = \frac{1}{n}\sum_{t\in\mathcal{T}}\sum_{y\neq y_t^*} n(y,\mathcal{X}_t;\mathcal{D}) \\
&= \sum_{t\in\mathcal{T}}\widehat{p}(\mathcal{X}_t)\sum_{y\neq y_t^*}\frac{n(y,\mathcal{X}_t;\mathcal{D})}{\sum_{y'\in\mathcal{Y}} n(y',\mathcal{X}_t;\mathcal{D})} = \sum_{t\in\mathcal{T}}\widehat{p}(\mathcal{X}_t)\sum_{y\neq y_t^*}\widehat{p}(y|\mathcal{X}_t).
\end{aligned}$$

From this the claim follows. □

This misclassification rate plays the role of the deviance loss $D^*(\boldsymbol{N},(\bar{\lambda}_t)_{t\in\mathcal{T}})$ given in (6.12). It estimates the probability that a randomly chosen case $(Y,\boldsymbol{x})\in\mathcal{D}$ is misclassified.

The above construction implicitly assumes that misclassification is equally weighted for all labels in $\mathcal{Y}$. If preference should be given to a certain misclassification, we may introduce another loss function with $L(y,y^*)\geq 0$ for $y^*\neq y\in\mathcal{Y}$, and $L(y,y)=0$ for all $y\in\mathcal{Y}$. We interpret $L(y,y^*)$ to be the loss of classifying a response as y^* instead of the "true" label y. In this case we replace class assignment rule (6.28) by

$$y_t^* = \operatorname*{argmin}_{y^*\in\mathcal{Y}} \sum_{y\in\mathcal{Y}} L(y,y^*)\widehat{p}(y|\mathcal{X}_t), \tag{6.31}$$

and the above misclassification rate is modified to

$$\mathcal{L}^{\text{is}}_L(\mathcal{D},(y_t^*)_{t\in\mathcal{T}}) = \sum_{t\in\mathcal{T}}\widehat{p}(\mathcal{X}_t)\sum_{y\in\mathcal{Y}} L(y,y_t^*)\widehat{p}(y|\mathcal{X}_t). \tag{6.32}$$

This latter loss $\mathcal{L}^{\text{is}}_L(\mathcal{D},(\cdot)_{t\in\mathcal{T}})$ is motivated by the fact that $(y_t^*)_{t\in\mathcal{T}}$ defined in (6.31) minimizes (6.32) on partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$. Note that the misclassification rate (6.30) is the special case given by the loss function $L(y,y^*)=\mathbb{1}_{\{y\neq y^*\}}$, we also refer to (2.32).
Similar to Corollary 6.5 we have the following corollary.

Corollary 6.15. *Consider for a partition $(\mathcal{X}_t)_{t\in\mathcal{T}}$ of $\mathcal{X}$ an additional SBS for a given $t\in\mathcal{T}$. This gives the new leaves $\mathcal{T}'=(\mathcal{T}\setminus\{t\})\cup\{t0,t1\}$. We have $\mathcal{L}^{\text{is}}_L(\mathcal{D},(y_t^*)_{t\in\mathcal{T}})\geq \mathcal{L}^{\text{is}}_L(\mathcal{D},(y_t^*)_{t\in\mathcal{T}'})$.*

Proof. Note that we always assume a non-trivial SBS, i.e. we require existence of $t\in\mathcal{T}$ such that $\mathcal{X}_t$ contains at least two elements with different features. Then, we have for this binary index t

$$\begin{aligned}
\widehat{p}(\mathcal{X}_t)\sum_{y\in\mathcal{Y}} L(y,y_t^*)\widehat{p}(y|\mathcal{X}_t) &= \sum_{y\in\mathcal{Y}} L(y,y_t^*)\widehat{p}(y,\mathcal{X}_t) \\
&= \sum_{y\in\mathcal{Y}} L(y,y_t^*)\widehat{p}(y,\mathcal{X}_{t0}) + \sum_{y\in\mathcal{Y}} L(y,y_t^*)\widehat{p}(y,\mathcal{X}_{t1}) \\
&\geq \widehat{p}(\mathcal{X}_{t0})\sum_{y\in\mathcal{Y}} L(y,y_{t0}^*)\widehat{p}(y|\mathcal{X}_{t0}) + \widehat{p}(\mathcal{X}_{t1})\sum_{y\in\mathcal{Y}} L(y,y_{t1}^*)\widehat{p}(y|\mathcal{X}_{t1}).
\end{aligned}$$

This completes the proof. □

This implies that in the SBS Tree Growing Algorithm on page 151 we may replace (6.11) by a similar condition measuring the decrease in misclassification rate

$$\Delta D^*_{\mathcal{X}_t}(\varsigma_t) = \mathcal{L}^{\text{is}}_L(\mathcal{D}, (y^*_t)_{t\in\mathcal{T}}) - \mathcal{L}^{\text{is}}_L(\mathcal{D}, (y^*_t)_{t\in\mathcal{T}'}) \geq 0, \tag{6.33}$$

using the notation of Corollary 6.15.

Conclusion. We conclude for the tree classification case that all the results of the tree regression case remain true if we replace in (6.11) the deviance loss improvement (6.10) by the misclassification rate improvement (6.33).

Impurity measures

In definition (6.30) we have naturally assumed that we would like to measure the misclassification rate. This misclassification rate has the disadvantage that the considered *impurity function*

$$\phi_R\left(\widehat{p}(0|\mathcal{X}_t), \ldots, \widehat{p}(J-1|\mathcal{X}_t)\right) \;=\; \sum_{y\in\mathcal{Y}} \mathbb{1}_{\{y\neq y^*_t\}} \widehat{p}(y|\mathcal{X}_t) \;=\; 1 - \widehat{p}(y^*_t|\mathcal{X}_t),$$

with y^*_t given by (6.28), is not differentiable in its arguments which might cause problems in optimization. In particular, ϕ_R contains y^*_t which is obtained by an optimization in the empirical probabilities $\widehat{p}$. For instance, for $J = 2$ we have on the 1-unit simplex (choose $p \in [0,1]$ and set $q = 1-p \in [0,1]$)

$$\phi_R(p,q) \;=\; \phi_R(p,1-p) \;=\; 1 - \max\{p, 1-p\},$$

which is not differentiable in $p = 1/2$. Therefore, this impurity function ϕ_R is often replaced by other impurity functions. The two most popular ones are the Gini index function

$$\phi_G\left(\widehat{p}(0|\mathcal{X}_t), \ldots, \widehat{p}(J-1|\mathcal{X}_t)\right) \;=\; \sum_{y\in\mathcal{Y}} \left(1 - \widehat{p}(y|\mathcal{X}_t)\right) \widehat{p}(y|\mathcal{X}_t),$$

and the entropy function

$$\phi_E\left(\widehat{p}(0|\mathcal{X}_t), \ldots, \widehat{p}(J-1|\mathcal{X}_t)\right) \;=\; \sum_{y\in\mathcal{Y}} -\log\left(\widehat{p}(y|\mathcal{X}_t)\right) \widehat{p}(y|\mathcal{X}_t).$$

These two impurity functions have the advantage that they are differentiable. For instance for $J = 2$ we have on the 1-unit simplex (for $q = 1-p \in [0,1]$)

$$\phi_G(p,q) \;=\; \phi_G(p,1-p) \;=\; 2p(1-p),$$

and

$$\phi_E(p,q) \;=\; -p\log p - (1-p)\log(1-p),$$

see also Figure 6.7. These considerations motivate to replace misclassification rate (6.30),

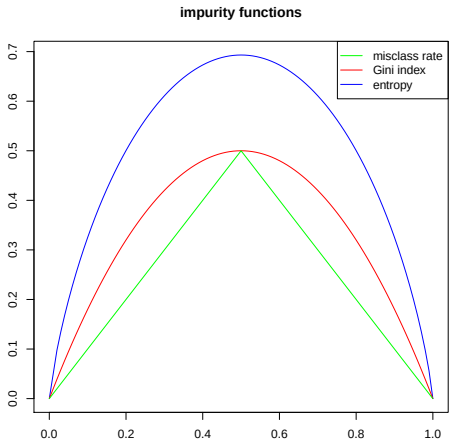


Figure 6.7: Impurity functions: misclassification error rate ϕ_R, Gini index function ϕ_G, entropy function ϕ_E for $J = 2$ as a function of $p \in [0, 1]$.

see also Lemma 6.14, by the impurity measures

$$\mathcal{L}^{\text{is}}_{\diamond}(\mathcal{D}, (\mathcal{X}_t)_{t\in\mathcal{T}}) = \sum_{t\in\mathcal{T}} \widehat{p}(\mathcal{X}_t)\ \phi_{\diamond}\left(\widehat{p}(0|\mathcal{X}_t), \ldots, \widehat{p}(J-1|\mathcal{X}_t)\right), \tag{6.34}$$

for $\diamond \in \{G, E\}$. We can use this impurity measure as objective function.

Choose a binary index $t \in \mathcal{T}$ and consider a SBS $\varsigma_t = (\mathcal{X}_{t0}, \mathcal{X}_{t1})$ of $\mathcal{X}_t$. This SBS leads to a change in impurity given by

$$\Delta D^*_{\mathcal{X}_t}(\varsigma_t) = \mathcal{L}^{\text{is}}_{\diamond}(\mathcal{D}, (\mathcal{X}_t)_{t\in\mathcal{T}}) - \mathcal{L}^{\text{is}}_{\diamond}(\mathcal{D}, (\mathcal{X}_t)_{t\in\mathcal{T}'}), \tag{6.35}$$

where $\mathcal{T}'$ denotes the resulting leaves also considering the additional binary split ς_t (similar to Corollary 6.15).

Remarks.

- Note that these two impurity functions ϕ_G and ϕ_E no longer depend on the class assignments $(y^*_t)_{t\in\mathcal{T}}$. Therefore, the class assignment (6.28) is done *after* the SBS Tree Growing Algorithm has been performed.
- ϕ_G and ϕ_E are called impurity functions because they measure the impurity on the $(J-1)$-unit simplex. They take the maximum in the center $(1/J, \ldots, 1/J)$, the minimum in the corners $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$ and are Schur concave.
- By applying the SBS Tree Growing Algorithm with criterion (6.35) in (6.11) we improve the purity on the resulting leaves in each step of the algorithm, however, in terms of the misclassification rate there might be better SBS. On the other hand, misclassification rates are not always sufficiently sensitive in SBS and the Gini index or the entropy function may have a better behavior on the final result.

- Pruning and cross-validation can then be done considering misclassification rates.
- The entropy impurity measure is closely related to the deviance loss function of a categorical distribution.

Appendix to Chapter 6

6.4 Proofs of pruning results

In this section we prove the results on cost-complexity pruning.

Proof of Theorem 6.9. This theorem is proved in Theorem 10.7 of [16]. Choose $\eta \geq 0$ fixed and assume that $\mathsf{T}' \subset \mathsf{T}$ describes a non-trivial binary (sub-)tree of T having identical root node $t = 0$. We decouple this binary tree T' into its root tree $\mathsf{T}'_{(-0)} = \{0\}$ and the binary subtrees $\mathsf{T}'_{(00+)}$ and $\mathsf{T}'_{(01+)}$ having new root nodes with binary indexes 00 and 01, respectively, see also (6.14). Observe that this provides a natural partition of the leaves of T' with leaf indexes $\mathcal{T}' = \mathcal{T}'_{(00+)} \cup \mathcal{T}'_{(01+)}$. This implies for the cost-complexity measure of the tree T'

$$\begin{aligned} R_\eta(\mathsf{T}') &= D^*(\boldsymbol{N}, (\bar{\lambda}_t)_{t\in\mathcal{T}'}) + \eta|\mathcal{T}'| = \sum_{t\in\mathcal{T}'} \left(D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) + \eta\right) \\ &= \sum_{t\in\mathcal{T}'_{(00+)}} \left(D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) + \eta\right) + \sum_{t\in\mathcal{T}'_{(01+)}} \left(D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) + \eta\right) \qquad (6.36) \\ &= R_\eta(\mathsf{T}'_{(00+)}) + R_\eta(\mathsf{T}'_{(01+)}), \end{aligned}$$

where the last identity denotes for $\tau = 0, 1$

$$R_\eta(\mathsf{T}'_{(0\tau+)}) = \sum_{t\in\mathcal{T}'_{(0\tau+)}} D^*_{\mathcal{X}_t}\left(\boldsymbol{N}, \bar{\lambda}_t\right) + \eta|\mathcal{T}'_{(0\tau+)}|.$$

Note that we use a slight abuse of notation here because the trees $\mathsf{T}'_{(0\tau+)}$ provide a partition of $\mathcal{X}_{0\tau} = (\mathcal{X}_t)_{t\in\mathcal{T}'_{(0\tau+)}}$, and not of the total feature space $\mathcal{X}$. But observe that $(\mathcal{X}_{0\tau})_{\tau=0,1}$ is a partition of the total feature space $\mathcal{X}$.

Identity (6.36) shows that finding minimizers $\mathsf{T}'(\eta)$ of (6.15) can be done inductively by finding minimizers (independently) on the sub-trees $\mathsf{T}_{(00+)}$ and $\mathsf{T}_{(01+)}$ of T. Thus, (6.36) reduces the number of generations by one digit and iterating this provides the proof. The formal completion of the proof is obtained by an explicit construction of the smallest cost-optimal binary subtree using that (6.36) can be applied to any split in any generation and using the fact that there are only finitely many binary subtrees. □

Proof of Theorem 6.11. Choose a finite binary tree T. This finite binary tree has at most finitely many subtrees T' that have the same root node $t = 0$. Therefore, the minimum in (6.17) is well-defined for all $\eta \geq 0$. For any subtree T' of T the function (6.16) is positive, strictly increasing and linear. Therefore, the minimum in (6.17) over finitely many such positive and strictly increasing linear functions needs to be positive, strictly increasing, piece-wise linear and concave with at most finitely many values $0 < \eta_1 < \ldots < \eta_\kappa < \infty$ where it is not differentiable, i.e. where the linear functions (6.16) intersect for different subtrees T'. We additionally set $\eta_0 = 0$ and $\eta_{\kappa+1} = \infty$. The smallest cost-optimal binary subtree of (6.15) in η_ℓ is given by $\mathsf{T}^{(\ell)} = \mathsf{T}(\eta_\ell)$, for $\ell = 0, \ldots, \kappa$, see Corollary 6.10. This implies that

$$r(\eta_\ell) = \min_{\mathsf{T}'\subset\mathsf{T}} r_{\mathsf{T}'}(\eta_\ell) = \min_{\mathsf{T}'\subset\mathsf{T}} R_{\eta_\ell}(\mathsf{T}') = R_{\eta_\ell}(\mathsf{T}(\eta_\ell)) = R_{\eta_\ell}(\mathsf{T}^{(\ell)}).$$

Since on the open interval $(\eta_\ell, \eta_{\ell+1})$ no other linear function $r_{\mathsf{T}'}(\cdot)$ intersects the straight line $r_{\mathsf{T}^{(\ell)}}(\cdot)$ it follows that $r(\eta) = r_{\mathsf{T}^{(\ell)}}(\eta)$ on $[\eta_\ell, \eta_{\ell+1})$.

Finally, we need to show that the sequence of the smallest cost-optimal binary subtrees $(\mathsf{T}^{(\ell)})_{\ell=0,\ldots,\kappa}$ is decreasing for increasing η_ℓ. Pursuing the last argument to the point $\eta_{\ell+1}$, we see from the continuity of the functions $r_{\mathsf{T}'}(\cdot)$ that $R_{\eta_{\ell+1}}(\mathsf{T}(\eta_\ell)) = R_{\eta_{\ell+1}}(\mathsf{T}(\eta_{\ell+1}))$. Since $\mathsf{T}(\eta_{\ell+1})$ is the smallest cost-optimal binary subtree in $\eta_{\ell+1}$ we have that $\mathsf{T}(\eta_\ell) \supset \mathsf{T}(\eta_{\ell+1})$ and $\mathsf{T}(\eta_\ell)$ can be considered as a minimizing tree $\mathsf{T}'(\eta_{\ell+1})$ in $\eta_{\ell+1}$ that is not smallest in the sense of as described by Theorem 6.9. Finally, the root tree $\{0\}$ is the unique binary subtree with the smallest slope $|\mathcal{T}'| = 1$, therefore for $\eta \to \infty$ the root tree is cost-complexity optimal and $\mathsf{T}^{(\kappa)} = \{0\}$. This finishes the proof. □

Proof of Lemma 6.12. To initialize we distinguish $\ell = 0$ and $\ell > 0$. We start with $\ell = 0$. Choose $t \in \mathsf{T} \setminus \mathcal{T}$. For every SBS considered in the SBS Tree Growing Algorithm we have a real improvement in the Poisson deviance loss which implies that

$$D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}}) < D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}_{(-t)}}).$$

This implies that $R_0(\mathsf{T}) < R_0(\mathsf{T}_{(-t)})$ and therefore provides the initialization $\mathsf{T}^{(0)} = \mathsf{T}$ for regularization parameter $\eta = \eta_0 = 0$, i.e.

$$R_{\eta_0}(\mathsf{T}^{(0)}) < R_{\eta_0}(\mathsf{T}^{(0)}_{(-t)}) \qquad \text{for any } t \in \mathsf{T}^{(0)} \setminus \mathcal{T}^{(0)}.$$

For $\ell > 0$ we know by construction that

$$R_{\eta_\ell}(\mathsf{T}^{(\ell)}) < R_{\eta_\ell}(\mathsf{T}^{(\ell)}_{(-t)}) \qquad \text{for any } t \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)},$$

because $\mathsf{T}^{(\ell)} = T(\eta_\ell) \neq \{0\}$ is the smallest cost-optimal subtree that minimizes the cost-complexity measure for regularization constant η_ℓ.
For any $t \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ we have $|\mathcal{T}^{(\ell)}| > |\mathcal{T}^{(\ell)}_{(-t)}|$. This implies that for any $t \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$ there exists an $\eta(t) > \eta_\ell$ with

$$R_{\eta(t)}(\mathsf{T}^{(\ell)}) = r_{\mathsf{T}^{(\ell)}}(\eta(t)) = r_{\mathsf{T}^{(\ell)}_{(-t)}}(\eta(t)) = R_{\eta(t)}(\mathsf{T}^{(\ell)}_{(-t)}),$$

because the linear function $r_{\mathsf{T}^{(\ell)}}(\cdot)$ has a bigger slope than the linear function $r_{\mathsf{T}^{(\ell)}_{(-t)}}(\cdot)$. This $\eta(t) > \eta_\ell$ is given by the solution of

$$\begin{aligned} 0 &= r_{\mathsf{T}^{(\ell)}}(\eta(t)) - r_{\mathsf{T}^{(\ell)}_{(-t)}}(\eta(t)) \\ &= D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}}) - D^*(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(-t)}}) + \eta(t)|\mathcal{T}^{(\ell)}| - \eta(t)|\mathcal{T}^{(\ell)}_{(-t)}| \\ &= D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(t+)}}) - D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\{t\}}) + \eta(t)\left(|\mathcal{T}^{(\ell)}_{(t+)}| - 1\right), \end{aligned} \tag{6.37}$$

and thus

$$\eta(t) = \frac{D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\{t\}}) - D^*_{\mathcal{X}_t}(\boldsymbol{N}, (\bar\lambda_s)_{s\in\mathcal{T}^{(\ell)}_{(t+)}})}{|\mathcal{T}^{(\ell)}_{(t+)}| - 1}.$$

The weakest node $t^*_{\ell+1}$ is then given by the minimal $\eta(t)$ where the minimum is considered for all nodes $t \in \mathsf{T}^{(\ell)} \setminus \mathcal{T}^{(\ell)}$. This determines the first intersection after η_ℓ of $r_{\mathsf{T}^{(\ell)}}(\cdot)$ with any other function $r_{\mathsf{T}^{(\ell)}_{(-t)}}(\cdot)$, and therefore determines $\eta_{\ell+1}$ and $\mathsf{T}^{(\ell+1)} = \mathsf{T}^{(\ell)}_{(-t^*_{\ell+1})}$. □

Chapter 7

Ensemble Learning Methods

In this chapter we present estimation methods that are based on combining several estimators in different ways, for instance, by averaging or stage-wise adaptive learning. Combining multiple learning methods to one estimator is often called ensemble learning.

7.1 Bootstrap simulation

The bootstrap simulation method goes back to Efron [35] and Efron–Tibshirani [38]. The main idea behind bootstrap is to simulate from an estimated model in order to gain more insight. This can either be done in a parametric or in a non-parametric way. The presentation in this section is taken from the lecture notes of Bühlmann–Mächler [20], Chapter 5, and it is similar to Section 4.3 in Wüthrich–Merz [141].

7.1.1 Non-parametric bootstrap

We start from i.i.d. observations $Y_1, \ldots, Y_n$ coming from an unknown distribution function F. Based on these observations we may construct an estimator

$$\widehat{\theta}_n = g(Y_1, \ldots, Y_n), \tag{7.1}$$

of a (real-valued, unknown but well-defined) quantity θ of interest. The function $g(\cdot)$ is known and we would like to understand its properties as a function of random variables $Y_1, \ldots, Y_n$. For instance, we may want to analyze the distributional properties of $\widehat{\theta}_n$, i.e. for measurable sets A we would like to study

$$\mathbb{P}\left[\widehat{\theta}_n \in A\right] = \mathbb{P}\left[g(Y_1, \ldots, Y_n) \in A\right] = \int \mathbb{1}_{\{g(y_1,\ldots,y_n)\in A\}} dF_{\boldsymbol{Y}}(y_1, \ldots, y_n), \tag{7.2}$$

where $F_{\boldsymbol{Y}}$ is the joint (product) distribution function of the i.i.d. observations $Y_i \sim F$. Since F is not known, the distributional properties of $\widehat{\theta}_n$ cannot be determined. Bootstrap replaced the unknown distribution F by the empirical distribution of the data

$$\widehat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{Y_i \le x\}},$$

and to simulate from this empirical distribution. This provides the following non-parametric bootstrap algorithm.

(Non-Parametric) Bootstrap Algorithm.

(1) Repeat for $m = 1, \ldots, M$

(a) simulate i.i.d. observations $Y_1^*, \ldots, Y_n^*$ from $\widehat{F}_n$;

(b) calculate the estimator

$$\widehat{\theta}_n^{(m*)} = g(Y_1^*, \ldots, Y_n^*).$$

(2) Return $\widehat{\theta}_n^{(1*)}, \ldots, \widehat{\theta}_n^{(M*)}$ and the corresponding bootstrap distribution

$$F^*_{\widehat{\theta}_n}(\vartheta) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}_{\{\widehat{\theta}_n^{(m*)} \le \vartheta\}}.$$

We may now use the bootstrap distribution $F^*_{\widehat{\theta}_n}$ as an estimate of the distribution of $\widehat{\theta}_n$, that is, in view of (7.2) we estimate

$$\widehat{\mathbb{P}}\left[\widehat{\theta}_n \in A\right] \;=\; \mathbb{P}^*\left[\widehat{\theta}_n \in A\right] \;=\; \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}_{\{\widehat{\theta}_n^{(m*)} \in A\}}. \tag{7.3}$$

Remarks.

- Strictly speaking there is an intermediate step in (7.3) that is highlighted in Section 4.3.1 of [141], namely, the quality of approximation (7.3) for (7.2) depends on two things. Firstly, on the richness of the observations $Y_1, \ldots, Y_n$, because

$$\mathbb{P}^*\left[\widehat{\theta}_n \in A\right] = \mathbb{P}^*_{\{Y_1,\ldots,Y_n\}}\left[\widehat{\theta}_n \in A\right].$$

Thus, the bootstrap probability $\mathbb{P}^* = \mathbb{P}^*_{\{Y_1,\ldots,Y_n\}}$ is a conditional probability, given the data $Y_1, \ldots, Y_n$. Secondly, it depends on M and the random drawings from $\widehat{F}_n$, however, this source of uncertainty can be controlled by the law of large numbers.

- The bootstrap distribution can be used to estimate, for instance, the mean of the estimator $\widehat{\theta}_n$ given in (7.1) which is set as

$$\widehat{\mathbb{E}}\left[\widehat{\theta}_n\right] = \mathbb{E}^*\left[\widehat{\theta}_n\right] = \frac{1}{M} \sum_{m=1}^{M} \widehat{\theta}_n^{(m*)},$$

and similarly for its variance

$$\widehat{\mathrm{Var}}\left(\widehat{\theta}_n\right) = \mathrm{Var}^*\left(\widehat{\theta}_n\right) = \frac{1}{M} \sum_{m=1}^{M} \left(\widehat{\theta}_n^{(m*)} - \mathbb{E}^*\left[\widehat{\theta}_n\right]\right)^2.$$

- We can then ask various questions about consistency, quality of bootstrap approximation, etc. We refer to Bühlmann–Mächler [20] for more details.

- Basically, the bootstrap algorithm is based on the Glivenko–Cantelli [54, 22] theorem which says that for i.i.d. observations the empirical distribution $\widehat{F}_n$ converges uniformly to the true distribution function F, see Theorem 20.6 in Billingsley [11].

7.1.2 Parametric bootstrap

The parametric bootstrap differs from its non-parametric counterpart in the sense that we assume a given parametric distribution function $F = F_\theta$ with unknown parameter θ for the i.i.d. observations $Y_1, \ldots, Y_n$. We then use estimator

$$\widehat{\theta}_n = g(Y_1, \ldots, Y_n), \tag{7.4}$$

for the unknown (say, real-valued) parameter θ of interest.

(Parametric) Bootstrap Algorithm.

---

(1) Repeat for $m = 1, \ldots, M$

   (a) simulate i.i.d. observations $Y_1^*, \ldots, Y_n^*$ from $F_{\widehat{\theta}_n}$;

   (b) calculate the estimator

$$\widehat{\theta}_n^{(m*)} = g(Y_1^*, \ldots, Y_n^*).$$

(2) Return $\widehat{\theta}_n^{(1*)}, \ldots, \widehat{\theta}_n^{(M*)}$ and the corresponding bootstrap distribution

$$F^*_{\widehat{\theta}_n}(\vartheta) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}_{\{\widehat{\theta}_n^{(m*)} \le \vartheta\}},$$

   for the estimated distribution of $\widehat{\theta}_n$.

The remainder is equivalent to the non-parametric bootstrap and the same remarks apply.

Remark. The parametric bootstrap needs some care because the chosen distribution can be misspecified. For instance, if we assume that $Y_1, \ldots, Y_n$ follow a Poisson distribution with estimated claim frequency $\widehat{\lambda}$, then one should additionally check whether the data is not over-dispersed. If the data is over-dispersed and we work with a Poisson distribution, we will underestimate uncertainty. This is not the case in the non-parametric bootstrap as long as the data is a typical observation.

7.2 Bagging

Bagging goes back to Breiman [13]. It combines **b**ootstrap and **agg**regat**ing**. We apply bagging to the classification and regression trees (CART) constructed in Chapter 6. Two main disadvantages of standardized binary splits (SBS) in the CART construction are that they only lead to piece-wise constant estimates (because we apply rectangular splits) and that they can be rather unstable under slight changes in the observations, i.e. a small change in an observation may lead to a different split (possibly close to the root of the tree). This different split may result in a completely different tree. For these reasons one aims at constructing a whole family (ensemble) of tree estimators which should become more stable under averaging (and aggregating). In this section we generate this family of tree estimators by the bootstrap algorithm.

7.2.1 Aggregating

Before we describe bagging we would like to discuss aggregating (and averaging) of estimators. We do this with the explicit Poisson model at hand. For a given estimator $\widehat{\lambda}$ of the true frequency $\lambda^\star$ we have the following Poisson deviance loss

$$D^*(\boldsymbol{N}, \widehat{\lambda}) = 2 \sum_{i=1}^n -N_i + N_i \log N_i + \widehat{\lambda}(\boldsymbol{x}_i)v_i - N_i \log \left(\widehat{\lambda}(\boldsymbol{x}_i)v_i\right) \ \geq \ 0,$$

where $N_i \log N_i = 0$ for $N_i = 0$. For the true expected frequency $\lambda^\star$ we obtain expected average Poisson deviance loss over our portfolio $(\boldsymbol{x}_i, v_i)_{i=1,\ldots,n}$

$$\frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^\star)\right] = \frac{2}{n} \sum_{i=1}^n \mathbb{E}\left[N_i \log N_i\right] - \lambda^\star(\boldsymbol{x}_i)v_i \log\left(\lambda^\star(\boldsymbol{x}_i)v_i\right) \geq 0,$$

where the last inequality follows because $y \mapsto y \log y$ is a convex function on $\mathbb{R}_+$. The single terms (expected values) of this sum are illustrated in Figure 1.1. This average Poisson deviance loss can be compared to the one obtained from the estimated frequency $\widehat{\lambda}$. Assume that the assumptions of Proposition 1.8 are fulfilled for $\widehat{\lambda}$, and $\boldsymbol{N}$ describes an independent copy of the data $\mathcal{D}$ (which has been used to estimate λ). Then we have, see (1.12),

$$\frac{1}{n}\mathbb{E}\left[D^*(\boldsymbol{N}, \widehat{\lambda})\right] = \frac{1}{n}\mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^\star)\right] + \mathbb{E}\left[\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)\right], \tag{7.5}$$

with estimation loss estimate, see (1.13),

$$\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star) = \frac{1}{n} \sum_{i=1}^n 2v_i \left[\widehat{\lambda}(\boldsymbol{x}_i) - \lambda^\star(\boldsymbol{x}_i) - \lambda^\star(\boldsymbol{x}_i) \log\left(\frac{\widehat{\lambda}(\boldsymbol{x}_i)}{\lambda^\star(\boldsymbol{x}_i)}\right)\right] \ \geq \ 0,$$

the inequality follows similar to the one in Proposition 1.8. This provides the following corollary.

Corollary 7.1. *Under the above assumptions we have*

$$\frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \widehat{\lambda})\right] \ \geq \ \frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^\star)\right].$$

Next we construct an *aggregated estimator*. We therefore assume that we can construct i.i.d. estimators $\widehat{\lambda}^{(m)} \overset{(\mathrm{d})}{=} \widehat{\lambda}$, $m \geq 1$, also being independent of $\boldsymbol{N}$ and fulfilling the assumptions of Proposition 1.8. We define the aggregated estimator by

$$\bar{\lambda}^M_{\mathrm{agg}}(\cdot) = \frac{1}{M} \sum_{m=1}^M \widehat{\lambda}^{(m)}(\cdot). \tag{7.6}$$

We have already met this idea in (5.28).

Proposition 7.2. *Under the above assumptions we have*

$$\frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \widehat{\lambda})\right] \ \geq \ \frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \bar{\lambda}^M_{\mathrm{agg}})\right] \ \geq \ \frac{1}{n} \mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^\star)\right].$$

Proof. The second inequality follows similarly to Corollary 7.1. For the first inequality we have (using that $\widehat{\lambda}^{(m)}$ are i.i.d. estimators having the same distribution as $\widehat{\lambda}$)

$$\frac{1}{n}\,\mathbb{E}\left[D^*(\boldsymbol{N},\bar{\lambda}^M_{\text{agg}})\right] = \frac{1}{n}\,\mathbb{E}\left[D^*(\boldsymbol{N},\widehat{\lambda})\right] - \frac{2}{n}\,\sum_{i=1}^n \lambda^\star(\boldsymbol{x}_i)v_i\,\mathbb{E}\left[\log\left(\frac{\bar{\lambda}^M_{\text{agg}}(\boldsymbol{x}_i)}{\widehat{\lambda}(\boldsymbol{x}_i)}\right)\right].$$

Jensen's inequality implies

$$\mathbb{E}\left[\log\left(\frac{\bar{\lambda}^M_{\text{agg}}(\boldsymbol{x}_i)}{\widehat{\lambda}(\boldsymbol{x}_i)}\right)\right] \geq \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^M \log\widehat{\lambda}^{(m)}(\boldsymbol{x}_i)\right] - \mathbb{E}\left[\log\widehat{\lambda}(\boldsymbol{x}_i)\right] = 0.$$

This proves the claim. □

Remarks to Proposition 7.2.

- A crucial remark is that the average bias remains unchanged by aggregation (7.6), but the estimation variance is reduced as can be seen from Proposition 7.2 (this is basically explained by the law of large numbers and the central limit theorem). Thus, aggregation (and averaging) has the nice effect of reducing the estimation uncertainty. The only open question is how to construct i.i.d. estimators $\widehat{\lambda}^{(m)}$. This is discussed next.

- If additionally we know that $\widehat{\lambda}$ is unbiased for $\lambda^\star$, then the law of large numbers will imply that the aggregated estimator converges a.s. to the true $\lambda^\star$ as $M \to \infty$, and a central limit type theorem provides that the rate of convergence is of order $\sqrt{M}$. Additionally, a convergence result can be proved for the expected deviance losses. For more details we refer to Richman–Wüthrich [111].

7.2.2 Bagging for Poisson regression trees

Assume that we have Poisson observations $\mathcal{D}$, see (2.3), and that we have determined the regression tree estimator $\widehat{\lambda}(\boldsymbol{x})$ according to Sections 6.1 and 6.2. We can then use this estimated expected frequency to apply the parametric bootstrap algorithm of Section 7.1.2 to receive estimators for aggregating, i.e. we combine bootstrap and aggregating in the following algorithm.[1]

Bagging for Poisson Regression Trees.

---

(1) Repeat for $m = 1, \ldots, M$

   (a) simulate independent observations for $i = 1, \ldots, n$

$$N_i^* \;\sim\; \text{Poi}(\widehat{\lambda}(\boldsymbol{x}_i)v_i); \tag{7.7}$$

   (b) calculate the regression tree estimator $\boldsymbol{x} \mapsto \widehat{\lambda}^{(m*)}(\boldsymbol{x})$ from these bootstrap observations $\mathcal{D}^* = \{(N_1^*, \boldsymbol{x}_1, v_1), \ldots, (N_n^*, \boldsymbol{x}_n, v_n)\}$.

[1] Note that for the regression tree estimators $\widehat{\lambda}$ we use the credibility estimators (6.8). These are strictly positive if $\bar{\lambda}_0 > 0$.

(2) Return the bagging estimator

$$\boldsymbol{x} \;\mapsto\; \widehat{\lambda}^{M}_{\mathrm{Bag}}(\boldsymbol{x}) \;=\; \frac{1}{M}\sum_{m=1}^{M} \widehat{\lambda}^{(m*)}(\boldsymbol{x}).$$

Remarks.

- In the bagging algorithm above we assume that we always choose the same portfolio $(\boldsymbol{x}_i, v_i)_{i=1,\ldots,n}$. Of course, we could also resample the corresponding portfolio from its empirical distribution (drawing with replacements).
- The bagging estimator $\widehat{\lambda}^{M}_{\mathrm{Bag}}(\cdot)$ is a sample estimator approximation to $\widehat{\lambda}(\cdot)$, using the bootstrap samples $\widehat{\lambda}^{(m*)}(\cdot)$, $m = 1, \ldots, M$. In particular, we have

$$\widehat{\lambda}^{M}_{\mathrm{Bag}}(\cdot) = \widehat{\lambda}(\cdot) + \left(\widehat{\lambda}^{M}_{\mathrm{Bag}}(\cdot) - \widehat{\lambda}(\cdot)\right),$$

  where the second term is the bootstrap bias. It (usually) turns out that the bagging estimator has a smaller variance (compared to the original tree estimator) at the cost of an additional bias $\widehat{\lambda}^{M}_{\mathrm{Bag}}(\boldsymbol{x}) - \widehat{\lambda}(\boldsymbol{x})$. This is in the spirit of Proposition 7.2, however, we cannot do an exact statement here because we only simulate (bootstrap) from an estimated model.
- The Bagging for Poisson Regression Trees algorithm uses a parametric bootstrap in (7.7). A non-parametric version would use sampling with replacements directly from the data $\mathcal{D}$. The latter may be advantageous if the bias in $\widehat{\lambda}$ is too large (due to a tree that was chosen too small) or if the resulting regression structure shows too much over-dispersion in the data, i.e., if the regression structure cannot explain the data up to a noise term of unit variance.
- We conclude that bagging is mainly a variance reduction technique and, for the moment, it is not clear whether it also improves the Poisson deviance loss.

Example 7.3 (Aggregating and bagging)**.** We revisit the regression tree example considered in Table 6.4.

	$\mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^{\square})\right]/n$	$\mathbb{E}[\widehat{\mathcal{E}}(\lambda^{\square}, \lambda^{\star})]$
(1) true frequency $\lambda^{\square} = \lambda^{\star}$	27.7013	0.0000
(2) estimated frequency $\lambda^{\square} = \widehat{\lambda}$ from $\mathsf{T}^{\min}$	28.0774	0.3762
(3) estimated frequency $\lambda^{\square} = \bar{\lambda}^{5}_{\mathrm{agg}}$	27.9101	0.2088
(4) estimated frequency $\lambda^{\square} = \widehat{\lambda}^{5}_{\mathrm{Bag}}$	28.0645	0.3632
(5) estimated frequency $\lambda^{\square} = \widehat{\lambda}^{10}_{\mathrm{Bag}}$	28.0512	0.3499

Table 7.1: Bagging results for the synthetic example given in Appendix A; in 10^{-2}.

Line (1) of Table 7.1 gives the expected average Poisson deviance loss $\mathbb{E}\left[D^*(\boldsymbol{N}, \lambda^{\star})\right]/n$ w.r.t. the true frequency $\lambda^{\star}$ as chosen in Appendix A. This value is obtained numerically

by Monte Carlo simulation of $\boldsymbol{N}$ from the true model on our given portfolio, see Figure A.3 (rhs). We note that this expected value of $27.7013 \cdot 10^{-2}$ is slightly smaller than the one of the selected observation given by $27.7278 \cdot 10^{-2}$, see Table 6.4.

On line (2) of Table 7.1 we calculate the i.i.d. regression tree estimators $\widehat{\lambda}^{(m)}$ from i.i.d. samples $\boldsymbol{N}^{(m)}$, $m \geq 1$, using the true model of Appendix A. To construct the regression tree estimators we use exactly the same cost-complexity parameter $\mathtt{cp} = 8.217 \cdot 10^{-5}$ as has been used to receive the minimum cross-validation error tree $\mathsf{T}^{\min}$ on line (Ch6.2) of Table 6.4. This allows us to empirically calculate the estimation error $\mathbb{E}[\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)]$ of this regression tree estimator. The expected average Poisson deviance loss is then estimated from (7.5) and using line (1) of Table 7.1. The estimation error of $0.3762 \cdot 10^{-2}$ is very similar to the one of the selected sample given on line (Ch6.2) in Table 6.4. Thus, our tree $\mathsf{T}^{\min}$ seems to be a typical one. Moreover, the stratified 10-fold cross-validation error of $28.1388 \cdot 10^{-2}$ matches well $28.0774 \cdot 10^{-2}$ which expresses that cross-validation works properly here.

On line (3) of Table 7.1 we perform the same analysis as on line (2) except that we replace the i.i.d. regression tree estimators $\widehat{\lambda}^{(m)}$ by i.i.d. aggregated estimators $\bar{\lambda}^{5,k}_{\text{agg}} = \frac{1}{5}\sum_{m=1}^{5} \widehat{\lambda}^{((k-1)5+m)}$, $k \geq 1$, to estimate the aggregated figures w.r.t. $\bar{\lambda}^{5}_{\text{agg}}$ (for $M = 5$) empirically. As suggested by Proposition 7.2, we receive (clearly) better results by aggregation, in fact, the results are competitive with the neural network results, see Table 6.4. Unfortunately, this positive result is not of practical use, not knowing the true regression function $\lambda^\star$ we cannot sample i.i.d. estimators $\bar{\lambda}^{5,k}_{\text{agg}}$. Therefore, this method remains intractable.

Finally, lines (4)-(5) of Table 7.1 show the bagging estimators for $M = 5, 10$. To receive them, we replace the i.i.d. samples $\boldsymbol{N}^{(m)}$, $m \geq 1$, from the true regression function $\lambda^\star$ by i.i.d. boostrap samples $\boldsymbol{N}^{(m*)}$, $m \geq 1$, from the estimated regression function $\widehat{\lambda}$ (using the minimum cross-validation error tree $\mathsf{T}^{\min}$ of line (Ch6.2) of Table 6.4), see also (7.7) for bootstrapping. The remaining steps are done completely analogously to the steps on lines (3) and (2) of Table 7.1. Bagging leads to a small improvement in Poisson deviance loss compared to the tree estimator on line (2). However, the reduction is comparably small which suggests that the estimation error term is dominated by the estimation bias in the tree estimator $\widehat{\lambda}$ used for bootstrapping. ■

From the previous example we see that bagging leads in our example only to a small improvement (if at all). This may come from the fact that bootstrapping from the tree estimator $\widehat{\lambda}$ is too static, i.e. for every bootstrap sample we use the same origin and the same information (and hence the same (expected) bias). We conclude that bagging is not very helpful in our problem.

7.3 Random forests

Random forests are motivated by the fact that the bagging algorithm presented in the previous section is too static and it does not really lead to (much) improvement and bias reduction. This deficiency may be eliminated by growing a very large tree (which typically leads to a smaller bias) and then applying a more noisy bootstrap on this large tree. If we average over sufficiently many noisy bootstrap samples (similarly to bagging)

we should still get a small estimation variance. This is the basic idea behind random forests which goes back to Breiman [14].
We construct a very large tree T and a tree estimator $\widehat{\lambda}$, respectively, based on the original data $\mathcal{D}$. From this very large tree estimator we simulate (parametric) bootstrap samples according to (7.7). These bootstrap samples then serve as observations to construct binary regression trees. However, for each standardized binary split (SBS) decision we choose at random a fixed number $q^\star$ of feature components on which we base the split decisions. Typically, $1 \leq q^\star < q$, and as a consequence we may miss the optimal split because we do not consider all feature components.

POISSON RANDOM FOREST WITH $1 \leq q^\star \leq q$.

---

(0) Choose $1 \leq q^\star \leq q$ fixed.

(1) Repeat for $m = 1, \ldots, M$

(a) simulate independent observations for $i = 1, \ldots, n$

$$N_i^* \ \sim \ \mathrm{Poi}(\widehat{\lambda}(\boldsymbol{x}_i)v_i); \tag{7.8}$$

(b) calculate a SBS regression tree estimator $\boldsymbol{x} \mapsto \widehat{\lambda}^{(m,q^\star)}(\boldsymbol{x})$ from these bootstrap observations $\mathcal{D}^* = \{(N_1^*, \boldsymbol{x}_1, v_1), \ldots, (N_n^*, \boldsymbol{x}_n, v_n)\}$, where for each SBS decision in (6.11) and (6.13), respectively, we first choose at random $q^\star$ feature components of $\boldsymbol{x}$ on which the SBS split decision is based on.

(2) Return the random forest estimator

$$\boldsymbol{x} \ \mapsto \ \widehat{\lambda}_{\mathrm{RF}}^M(\boldsymbol{x}) \ = \ \frac{1}{M} \sum_{m=1}^{M} \widehat{\lambda}^{(m,q^\star)}(\boldsymbol{x}). \tag{7.9}$$

Remarks.

- Observe that the Poisson Random Forest algorithm differs from the Bagging for Poisson Regression Tree algorithm as soon as $q^\star < q$, because in this case we may miss the optimal split if it is not among the $q^\star$ feature components selected. This leads to more noisy trees and averaging smooths out this noise.
- Often, one chooses $q^\star \leq \sqrt{q}$ or $q^\star = q/3$.
- In (7.8) we use a parametric bootstrap, a non-parametric bootstrap would be achieved by using sampling with replacements directly from the observations.
- If the bias is still (too) large, one may choose a bigger tree. However, this question can typically not be answered in a satisfactory manner because usually the best tree model is not known.
- Choosing features at random may also help to break colinearity between feature components (decorrelate trees).

- Another interesting analysis is to study which feature component has been chosen how often in the random forest algorithm, and how much these splits decrease the deviance loss. This provides a measure of variable importance which may be illustrated in plots like Figure 17.5 in Efron–Hastie [37].

Example 7.4 (Random forests)**.** We continue our MTPL example considered in Table 6.4. The Poisson deviance loss version of the random forest algorithm is implemented in the R package `rfCountData`. The corresponding R code is given in Listing 7.1.

Listing 7.1: R command `rfPoisson`

```
1 library(rfCountData)
2
3 rf <- rfPoisson(x=dat[,c("age","ac","power","gas","brand","area","dens","ct")],
4                  offset=log(dat$expo), y=dat$claims,
5                  ntree=10, mtry=3, replace=TRUE, nodesize=1000)
6
7 print(rf)
8 plot(rf)
9 dat$predRF <- predict(rf, offset=log(dat$expo), newdata=dat)
```

Lines 3-4 specify the features $\boldsymbol{x}$, the years at risk v as log-offsets, and the responses N. `ntree` denotes the number M of bootstrapped trees, `mtry` gives the number $q^\star$ of feature components considered, and `replace` determines whether the drawing of cases should be done with our without replacements. Note that `rfPoisson` is based on a non-parametric bootstrap as it has been introduced by Breiman [14]. That is, (7.8) is replaced by drawings with or without replacements; on the non-selected samples the algorithm calculates an out-of-bag generalization loss, which is reported in Figure 7.1. Finally, `nodesize` determines the minimal number of cases in each leaf the tree estimators $\widehat{\lambda}^{(m,q^\star)}$ should contain.

In Table 7.2 we consider three different parametrizations of random forests: RF1 has $M = 10$ and `nodesize` $= 10'000$, RF2 has $M = 10$ and `nodesize` $= 1'000$, and RF3 has $M = 100$ and `nodesize` $= 1'000$. Thus, random forest RF1 is based on $M = 10$ comparably small tree estimators $\widehat{\lambda}^{(m,q^\star)}$ having in each leaf at least 10'000 cases. For random forest RF2 we decrease this number to 1'000 cases per leaf. From Figure 6.5 we know that these large trees tend to over-fit to the data. Averaging (7.9) should take care of this over-fitting, for RF2 we average over $M = 10$ tree estimators and for RF3 we average over $M = 100$ tree estimators.

From random forest estimate RF1 in Table 7.2 we observe that too small trees, i.e. too big values for `nodesize`, do not provide competitive models. However, increasing the size of the trees provides remarkably good predictive results. The two random forest estimates RF2 and RF3 are on a similar level as the neural network approaches in terms of estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$.

The difficulty with the large trees in RF2 and RF3 are the computational times. The construction of one such big tree takes roughly 60 seconds in the implementation of `rfPoisson`, thus, if averaging over $M = 100$ tree estimators takes roughly 1.5 hours.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.4) CANN1	28s	703	27.9292	27.9362	0.2284	27.8940	10.1577%
(Ch5.5) CANN2	27s	780	27.9306	27.9456	0.2092	27.8684	10.2283%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%
(Ch5.7) blended DNN	–	–	–	–	0.1336	27.6939	10.2691%
(Ch6.2) Tree2 $\mathsf{T}^{\min}$	65s	71	–	28.1388	0.3748	27.9156	10.2570%
(Ch7.1) RF1	60s	–	28.1375†	–	0.3642	28.0153	10.1901%
(Ch7.2) RF2	594s	–	28.2135†	–	0.2944	27.3573	10.0628%
(Ch7.3) RF3	5'931s	–	27.9808†	–	0.2256	27.2942	10.0863%

Table 7.2: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; † correspond to out-of-bag losses; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 6.4.

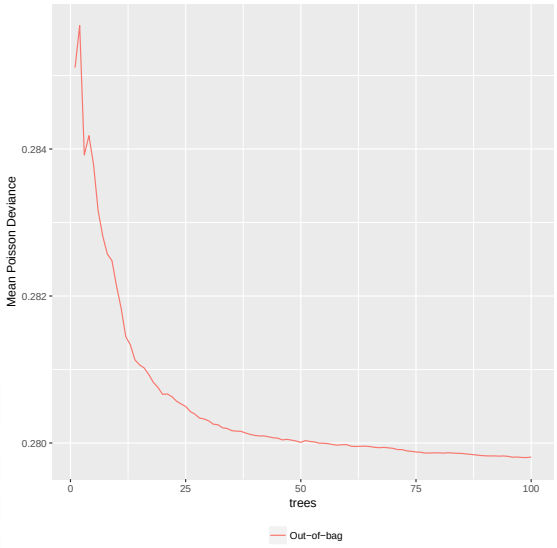


Figure 7.1: Decrease of out-of-bag losses in the random forest algorithm for trees of minimal leaf size of `nodesize` $= 1'000$.

In Figure 7.1 we plot the out-of-bag generalization losses on the not-selected samples in the bootstrap iterations. This plot suggests that the predictive model can further be improved by using more iterations M in the random forest algorithm. We conclude that random forests give powerful predictors, however, at rather high computational costs (in the current implementation). This finishes the example. ■

7.4 Boosting machines

Boosting is an iterative method that aims at combining many *weak predictions* into one powerful one. The underlying mechanism is completely different from bagging and random forests, namely, boosting aims at minimizing the in-sample loss by a stage-wise adaptive learning algorithm. Its roots go back to Valiant [128] and Kearns–Valiant [79, 80]. Schapire [118], Freund [44] and Freund–Schapire [45] popularized the method by presenting the AdaBoost algorithm and by providing rigorous results. This chapter is based on Chapter 10 of Hastie et al. [62].

7.4.1 Generic gradient boosting machine

Assume we are given an objective function $L(\cdot)$ and we aim at minimizing the in-sample loss over the (learning) sample $\mathcal{D}$. That is, we try to solve

$$\widehat{f} \;=\; \underset{f}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L\left(Y_i, f(\boldsymbol{x}_i), v_i\right), \tag{7.10}$$

where we restrict this optimization over a well-specified family of functions f. Observe that optimization (7.10) *only* specifies the function $f : \mathcal{X} \to \mathbb{R}$ in the feature values $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$. This implies that the above optimization problem can be replaced by trying to find optimal parameters $\boldsymbol{f} = (f_1, \ldots, f_n)' = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))' \in \mathbb{R}^n$ that minimize the in-sample loss

$$\mathcal{L}_L^{\text{is}}(\boldsymbol{f}) = \frac{1}{n}\sum_{i=1}^{n} L\left(Y_i, f_i, v_i\right) = \frac{1}{n}\sum_{i=1}^{n} L\left(Y_i, f(\boldsymbol{x}_i), v_i\right). \tag{7.11}$$

The saturated model would provide a minimal in-sample loss, but the saturated model leads to over-fitting. This is exactly the reason for introducing smoothness and regularity conditions on f in Chapter 3 on GAMs, see (3.7)-(3.8), where we restrict to natural cubic splines with multiplicative interactions (by using the log-link) for f.

In general, a non-trivial global solution to (7.10) is not feasible. Here, we solve the problem by restricting to another specific class of predictors (and functions), namely, we will consider regression tree functions with a given small number of leaves. Therefore, we design an algorithm that locally improves the situation in each iteration. This is exactly what the gradient boosting machine (GBM) is designed for. The GBM was developed in Friedman [46, 47], we also refer to Ridgeway [112] that serves the purpose of describing the corresponding implementation in R of the package `gbm`.

In a nutshell, assume that we have found a minimizer $\widehat{f}_{m-1}(\cdot)$ w.r.t. (7.10), where the minimization has been performed over a (small) family of admissible functions. In a next step, we may try to adaptively improve this estimator by considering the optimization problem

$$\widehat{g}_m \;=\; \underset{g \in \mathcal{G}_m}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L\left(Y_i, \widehat{f}_{m-1}(\boldsymbol{x}_i) + g(\boldsymbol{x}_i), v_i\right), \tag{7.12}$$

where we restrict this optimization to sufficiently simple functions $\mathcal{G}_m \supset \{g \equiv 0\}$. This then provides an improved estimator $\widehat{f}_m = \widehat{f}_{m-1} + \widehat{g}_m$. Iteration of these optimizations (7.12) for $m \geq 1$ will stage-wise adaptively improve an initial (weak) learner $\widehat{f}_0$. This

idea has already been presented in Section 5.1.4 on the CANN approach where we have started from a GLM estimator. This GLM estimator $\widehat{f}_0$ has been boosted in the sense of (7.12) by allowing for neural network features in $\widehat{g}_1$.

Recall that optimization (7.10) is only specified in the observed features $\boldsymbol{x}_i$, see (7.11). Assume that in the $(m-1)$-st step of the optimization algorithm we have found values in $\boldsymbol{x}_i$ given by $\boldsymbol{f}_{m-1} = (f_{m-1,1}, \ldots, f_{m-1,n})' \in \mathbb{R}^n$ that provide in-sample loss

$$\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) = \frac{1}{n}\sum_{i=1}^n L\left(Y_i, f_{m-1,i}, v_i\right).$$

In the next step, we perturb $\boldsymbol{f}_{m-1}$ locally so that it leads to a maximal local decrease in in-sample loss. This provides the next value $\boldsymbol{f}_m$ of the algorithm. Assume that the loss function $L(\cdot)$ is sufficiently smooth. The gradient $\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})$ indicates the locally biggest decrease in in-sample loss. This gradient is given by

$$\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) = \frac{1}{n}\left(\frac{\partial L(Y_1, f, v_1)}{\partial f}\Big|_{f=f_{m-1,1}}, \ldots, \frac{\partial L(Y_n, f, v_n)}{\partial f}\Big|_{f=f_{m-1,n}}\right)' \in \mathbb{R}^n.$$

The first order Taylor expansion around $\boldsymbol{f}_{m-1}$ provides

$$\mathcal{L}_L^{\text{is}}\left(\boldsymbol{f}\right) = \mathcal{L}_L^{\text{is}}\left(\boldsymbol{f}_{m-1}\right) + \nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})'\left(\boldsymbol{f} - \boldsymbol{f}_{m-1}\right) + o(\|\boldsymbol{f} - \boldsymbol{f}_{m-1}\|),$$

as $\|\boldsymbol{f} - \boldsymbol{f}_{m-1}\| \to 0$. If we choose a (small) step of size $\varrho > 0$ we have locally optimal first order update

$$\boldsymbol{f}_{m-1} \;\to\; \boldsymbol{f}_{m-1} - \varrho\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}). \tag{7.13}$$

This provides first order approximation to the in-sample loss in the updated estimate

$$\mathcal{L}_L^{\text{is}}\left(\boldsymbol{f}_{m-1} - \varrho\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})\right) = \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) - \varrho\left\|\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})\right\|^2 + o(\varrho), \tag{7.14}$$

as $\varrho \to 0$. Thus, we see that update (7.13) provides locally the biggest improvement in in-sample loss. This is completely analogous to the steepest gradient descent step given in (5.11). The optimal step size $\varrho_m > 0$ is then found by

$$\varrho_m \;=\; \underset{\varrho>0}{\operatorname{argmin}}\; \mathcal{L}_L^{\text{is}}\left(\boldsymbol{f}_{m-1} - \varrho\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})\right).$$

This provides locally optimal first order update

$$\boldsymbol{f}_{m-1} \;\to\; \boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \varrho_m\nabla\mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}),$$

and we iterate this algorithm. This update is exactly in the spirit of (7.12) and motivates the following generic GBM.

(Generic) Gradient Boosting Machine (GBM).

---

(0) Initialize the constant function $\widehat{f}_0(\boldsymbol{x}) = \varrho_0 = \underset{\varrho}{\operatorname{argmin}} \sum_{i=1}^n L\left(Y_i, \varrho, v_i\right)$.

(1) Repeat for $m = 1, \ldots, M$

(a) calculate the working responses $R_{m,i}$, $i = 1, \ldots, n$,

$$R_{m,i} = - \left. \frac{\partial L(Y_i, f, v_i)}{\partial f} \right|_{f=\widehat{f}_{m-1}(\boldsymbol{x}_i)};$$

(b) fit a regression model $\widehat{g}_m : \mathcal{X} \to \mathbb{R}$ to the working responses $(R_{m,i})_{i=1,\ldots,n}$ having corresponding features and volumes $(\boldsymbol{x}_i, v_i)_{i=1,\ldots,n}$;

(c) compute the optimal step size

$$\varrho_m \;=\; \underset{\varrho>0}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(Y_i, \widehat{f}_{m-1}(\boldsymbol{x}_i) + \varrho \widehat{g}_m(\boldsymbol{x}_i), v_i\right);$$

(d) update

$$\widehat{f}_{m-1}(\boldsymbol{x}) \;\to\; \widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \varrho_m \widehat{g}_m(\boldsymbol{x}). \tag{7.15}$$

(2) Return estimator $\boldsymbol{x} \mapsto \widehat{f}(\boldsymbol{x}) = \widehat{f}_M(\boldsymbol{x})$.

Remarks to the generic GBM.

- Step (1a): Observe that the scaled working responses $(R_{m,i}/n)_{i=1,\ldots,n}$ exactly correspond to the components of the negative gradient $-\nabla \mathcal{L}_L^{\text{is}}(\widehat{f}_{m-1}(\boldsymbol{x}_1), \ldots, \widehat{f}_{m-1}(\boldsymbol{x}_n))$. Thus, these working responses provide the locally optimal direction for the update.

- Step (1b): In the update (7.13) we optimize over all observed values $(f_{m-1,i})_i$. This is changed in step (1b) of the generic GBM for the following two reasons: (i) we would like to fit a function $\widehat{g}_m : \mathcal{X} \to \mathbb{R}$ that is defined on the entire feature space $\mathcal{X}$ (and not only in the observed feature values $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$); and (ii) we do not want over-fitting. Therefore, we rather fit a "lower-dimensional" regression model $\widehat{g}_m$ to these working responses (and we do not choose the saturated improvement). More illustration to this step is provided in the Poisson case below.

- Step (1d): We receive potential over-fitting if the number of iterations M in the generic GBM is chosen too large. To prevent from this kind of over-fitting, often a smaller step size is executed in (7.15): choose fixed shrinkage constant $\alpha \in (0, 1)$ and define the shrinkage updates

$$\widehat{f}_{m-1}(\boldsymbol{x}) \;\to\; \widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \alpha \varrho_m \widehat{g}_m(\boldsymbol{x}). \tag{7.16}$$

Regression step (1b) in the gradient boosting machine

Step (1b) of the GBM requires fitting a regression model $\widehat{g}_m : \mathcal{X} \to \mathbb{R}$ to the working responses $(R_{m,i}, \boldsymbol{x}_i, v_i)$, $i = 1, \ldots, n$. Observe that the optimal first order direction is, see (7.14),

$$-n \nabla \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) = (R_{m,1}, \ldots, R_{m,n})'.$$

We try to optimally approximate this direction by selecting the optimal regression function $\widehat{g}_m : \mathcal{X} \to \mathbb{R}$ within the family $\mathcal{G}_m$ of admissible regression functions on $\mathcal{X}$. The optimal L^2-distance approximation in step (1b) is given by

$$\widehat{g}_m \;=\; \underset{g_m \in \mathcal{G}_m}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(R_{m,i} - g_m(\boldsymbol{x}_i)\right)^2. \tag{7.17}$$

Thus, if we approximate direction $-n\nabla \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})$ by $\widehat{\boldsymbol{g}}_m = (\widehat{g}_m(\boldsymbol{x}_1), \ldots, \widehat{g}_m(\boldsymbol{x}_n))' \in \mathbb{R}^n$ we obtain for (7.14) approximation, as $\varrho \to 0$,

$$\begin{aligned} \mathcal{L}_L^{\text{is}}\left(\boldsymbol{f}_{m-1} + \varrho \widehat{\boldsymbol{g}}_m\right) &= \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) + \varrho \nabla \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1})' \widehat{\boldsymbol{g}}_m + o(\varrho) \\ &\approx \mathcal{L}_L^{\text{is}}(\boldsymbol{f}_{m-1}) - \varrho \left\|\widehat{\boldsymbol{g}}_m\right\|^2 / n. \end{aligned}$$

From this we see that the regression step (1b) results in finding the L^2-optimal regression function $\widehat{g}_m : \mathcal{X} \to \mathbb{R}$ in (7.17) that is close to the working responses. In particular, we use the square loss function (residual sum of squares) for this step of the generic GBM. For an other (more consistent) approach we refer to the next section on Poisson regression.

Remarks. The family $\mathcal{G}_m$ of admissible functions in (7.17) is often chosen to be the family of regression tree functions with a given small number of leaves. Having only a small number of leaves improves the regression function (7.15) in each stage-wise adaptive step weakly, and combining many weak learners typically leads to a powerful regression function $\widehat{f} = \widehat{f}_M$.

7.4.2 Poisson regression tree boosting machine

The generic GBM is a bit artificial in the Poisson case with the Poisson deviance loss as objective function. Steps (1a)-(1d) aim at locally improving the in-sample loss by *stage-wise adaptive* adding a new basis function $\widehat{g}_m$ to $\widehat{f}_{m-1}$. This is done via the gradient of the objective function so that the optimal first order Taylor expansion can be studied. In the following Poisson case we can consider these steps directly. Assume that the functions f play the role of the logged frequency $\log \lambda$. Suppose $\widehat{f}_{m-1}$ has been constructed and we consider the (stage-wise adaptive) in-sample loss optimization (set step size $\varrho = 1$)

$$\widehat{g}_m \;=\; \underset{g_m \in \mathcal{G}_m}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(Y_i, e^{\widehat{f}_{m-1}(\boldsymbol{x}_i) + g_m(\boldsymbol{x}_i)}, v_i\right), \tag{7.18}$$

for a given family $\mathcal{G}_m$ of admissible regression functions on $\mathcal{X}$. This in-sample loss is for the Poisson deviance loss function given by

$$\begin{aligned} D^*\left(\boldsymbol{N}, e^{\widehat{f}_{m-1} + g_m}\right) &= \sum_{i=1}^{n} 2N_i \left[\frac{v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i) + g_m(\boldsymbol{x}_i)}}{N_i} - 1 - \log\left(\frac{v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i) + g_m(\boldsymbol{x}_i)}}{N_i}\right)\right] \\ &= \sum_{i=1}^{n} 2N_i \left[\frac{w_i^{(m)} e^{g_m(\boldsymbol{x}_i)}}{N_i} - 1 - \log\left(\frac{w_i^{(m)} e^{g_m(\boldsymbol{x}_i)}}{N_i}\right)\right] \\ &= \sum_{i=1}^{n} L\left(Y_i, e^{g_m(\boldsymbol{x}_i)}, w_i^{(m)}\right), \end{aligned}$$

with *working weights* for $i = 1, \ldots, n$ given by

$$w_i^{(m)} = v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i)}. \tag{7.19}$$

From this we conclude that optimization problem (7.18) is in our Poisson case with deviance loss equivalent to solving

$$\widehat{g}_m \;=\; \underset{g_m \in \mathcal{G}_m}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(Y_i, e^{g_m(\boldsymbol{x}_i)}, w_i^{(m)}\right), \tag{7.20}$$

for a given family $\mathcal{G}_m$ of admissible regression functions on $\mathcal{X}$. Thus, in every step of the Poisson boosting machine we receive updated working weights $(w_i^{(m)})_{i=1,\ldots,n}$, playing the role of the volumes in the Poisson model, and replacing the role of the working responses in the boosting machine. This is not surprising because it is exactly what we have been using in (5.23)-(5.24) for boosting a GLM with neural network features in the Poisson case, that is, the logarithm of the working responses play the role of fixed offsets that are enhanced by a next regression model/boosting step.

This motivates iterative solutions of regression problems. For binary tree regressions this results in the following stage-wise adaptive learning algorithm. We set for the stage-wise adaptive improvements $\log \widehat{\mu}_m = \widehat{g}_m$.

Poisson Regression Tree Boosting Machine.

---

(0) Initialization:

  (a) choose $J \geq 2$ and $\alpha \in (0, 1]$ fixed;

  (b) set homogeneous overall MLE $\widehat{f}_0(\boldsymbol{x}) = \log \widehat{\lambda}_0(\boldsymbol{x}) = \log\left(\sum_{i=1}^n N_i / \sum_{i=1}^n v_i\right)$.

(1) Repeat for $m = 1, \ldots, M$

  (a) set working weights $w_i^{(m)}$ according to (7.19) and consider the working data

$$\mathcal{D}^{(m)} = \left\{\left(N_1, \boldsymbol{x}_1, w_1^{(m)}\right), \ldots, \left(N_n, \boldsymbol{x}_n, w_n^{(m)}\right)\right\};$$

  (b) construct a SBS Poisson regression tree estimator

$$\boldsymbol{x} \;\mapsto\; \widehat{\mu}_m(\boldsymbol{x}) = \sum_{t \in \mathcal{T}^{(m)}} \bar{\mu}_t^{(m)} \, \mathbb{1}_{\{\boldsymbol{x} \in \mathcal{X}_t^{(m)}\}}, \tag{7.21}$$

  with $|\mathcal{T}^{(m)}| = J$ leaves for the working data $\mathcal{D}^{(m)}$, see Section 6.1 and (6.1);

  (c) update the estimator

$$\begin{aligned} \widehat{f}_{m-1}(\boldsymbol{x}) \;\to\; \widehat{f}_m(\boldsymbol{x}) \;&=\; \widehat{f}_{m-1}(\boldsymbol{x}) + \alpha \widehat{g}_m(\boldsymbol{x}) \\ &=\; \widehat{f}_{m-1}(\boldsymbol{x}) + \alpha \sum_{t \in \mathcal{T}^{(m)}} \log(\bar{\mu}_t^{(m)}) \, \mathbb{1}_{\{\boldsymbol{x} \in \mathcal{X}_t^{(m)}\}}. \end{aligned} \tag{7.22}$$

(2) Return the expected frequency estimator

$$\boldsymbol{x} \quad \mapsto \quad \widehat{\lambda}(\boldsymbol{x}) = \exp\left\{\widehat{f}_M(\boldsymbol{x})\right\}.$$

Remarks on the Poisson Regression Tree Boosting Machine.

- In steps (1a)-(1c) we directly study the optimal local improvement (without considering an approximation to the first order Taylor expansion). Local is meant here in the sense that we disturb the previous solution $\widehat{f}_{m-1}(\cdot)$, which is included in the working weights $(w_i^{(m)})_{i=1,\ldots,n}$.
- In step (1b) we choose a SBS regression tree estimator for updating the fit. Of course, we could choose any other regression method here. Importantly, we fix the number of leaves $J \geq 2$ in advance, i.e. we do not aim for the optimal tree here. This J should not be chosen too large for several reasons:
  - speed of calculations (in each iteration $m = 1, \ldots, M$),
  - over-fitting, i.e. in general we are only looking for small or moderate improvements in each step, otherwise we obtain over-fitting and too wildly looking functions. Over-fitting can also be tuned by choosing an appropriate shrinkage constant $\alpha < 1$.
- The boosting machine is of particular interest for model back-testing. If we have an existing model $\widehat{\lambda}_m(\cdot) = \exp\{\widehat{f}_m(\cdot)\}$, then the boosting step analyzes whether we should additionally scale this model with a function $\widehat{\mu}_{m+1}(\cdot)$ different from 1. This idea is illustrated in more detail in a mortality modeling example in Deprez et al. [32].
- The Poisson regression tree boosting machine can be generalized to other distributions such the exponential dispersion family, see Lee–Lin [86].
- The SBS Poisson regression tree estimator in step (1b) is called *exact greedy algorithm* because it looks for the best possible split among all SBSs. It has been noticed that this can be very time consuming, in particular, if we have many categorical feature components (we also refer to the run times of the random forests in Table 7.2). Meanwhile there are other algorithms that, in particular, can handle categorical variables and sparse data (from one-hot encoding) more efficiently. A very powerful algorithm is XGBoost developed by Chen–Guestrin [26] that overcomes several issues, we refer to Ferrario–Hämmerli [42].

Example 7.5 (trees of depth 1)**.** We consider the Poisson regression tree boosting machine with $J = 2$ leaves. That is, every stage-wise adaptive learner in (7.21) has exactly one SBS $\varsigma_0 = (\mathcal{X}_{00}^{(m)}, \mathcal{X}_{01}^{(m)})$ of the entire feature space $\mathcal{X}_0 = \mathcal{X}$. Thus, in view of (6.4) we consider for each iteration $m = 1, \ldots, M$ the optimizations

$$\min_{\varsigma_0=(\mathcal{X}_{00}^{(m)},\mathcal{X}_{01}^{(m)})} D^*_{\mathcal{X}_{00}^{(m)}}\left(\boldsymbol{N}, \bar{\mu}_{00}^{(m)}\right) + D^*_{\mathcal{X}_{01}^{(m)}}\left(\boldsymbol{N}, \bar{\mu}_{01}^{(m)}\right),$$

where for $\tau = 0, 1$ we have MLEs

$$\bar{\mu}_{0\tau}^{(m)} = \frac{\sum_{i:\, \boldsymbol{x}_i \in \mathcal{X}_{0\tau}^{(m)}} N_i}{\sum_{i:\, \boldsymbol{x}_i \in \mathcal{X}_{0\tau}^{(m)}} w_i^{(m)}} = \frac{\sum_{i:\, \boldsymbol{x}_i \in \mathcal{X}_{0\tau}^{(m)}} N_i}{\sum_{i:\, \boldsymbol{x}_i \in \mathcal{X}_{0\tau}^{(m)}} v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i)}},$$

and the corresponding Poisson deviance losses are given by

$$D^*_{\mathcal{X}^{(m)}_{0\tau}}\left(\boldsymbol{N}, \bar{\mu}^{(m)}_{0\tau}\right) \;=\; \sum_{i:\, \boldsymbol{x}_i \in \mathcal{X}^{(m)}_{0\tau}} 2N_i \left[\frac{\bar{\mu}^{(m)}_{0\tau} v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i)}}{N_i} - 1 - \log\left(\frac{\bar{\mu}^{(m)}_{0\tau} v_i e^{\widehat{f}_{m-1}(\boldsymbol{x}_i)}}{N_i}\right)\right].$$

The optimal SBS $\varsigma_0 = (\mathcal{X}^{(m)}_{00}, \mathcal{X}^{(m)}_{01})$ selects one feature component x_l of $\boldsymbol{x}$ for this split and, thus, adds one "rectangular" split to $\mathcal{X}$. In Figure 7.2 we illustrate these rectangular splits for $M = 3$ iterations on a feature space $\mathcal{X} \subset [-1, 1]^2$ of dimension $q = 2$.

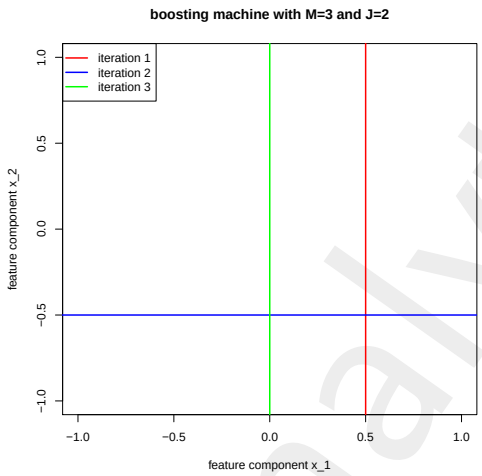


Figure 7.2: Poisson regression tree boosting with $J = 2$ illustrating $M = 3$ iterations.

Let us assume that the optimal SBS in iteration m is w.r.t. the (continuous) feature component $x_{l^*_m}$. Moreover, assume that it corresponds to the split question $x_{l^*_m} \leq c^*_m$, see Section 6.1.2. This then implies that we obtain multiplicative structure

$$\widehat{\lambda}_m(\boldsymbol{x}) \;=\; e^{\widehat{f}_m(\boldsymbol{x})} \;=\; \widehat{\lambda}_{m-1}(\boldsymbol{x}) \left(\bar{\mu}^{(m)}_{00}\; \mathbb{1}_{\{x_{l^*_m} \leq c^*_m\}} + \bar{\mu}^{(m)}_{01}\; \mathbb{1}_{\{x_{l^*_m} > c^*_m\}}\right).$$

From this we see that in the special case $J = 2$ we obtain multiplicative correction factors which depend on one single feature component only. Thus, we have

$$\begin{aligned}
\widehat{\lambda}_M(\boldsymbol{x}) \;&=\; \widehat{\lambda}_0(\boldsymbol{x}) \prod_{m=1}^{M} \left(\bar{\mu}^{(m)}_{00}\; \mathbb{1}_{\{x_{l^*_m} \leq c^*_m\}} + \bar{\mu}^{(m)}_{01}\; \mathbb{1}_{\{x_{l^*_m} > c^*_m\}}\right) \\
&=\; \widehat{\lambda}_0(\boldsymbol{x}) \prod_{m=1}^{M} \prod_{l=1}^{q} \exp\left\{\left(\log(\bar{\mu}^{(m)}_{00}) \mathbb{1}_{\{x_l \leq c^*_m\}} + \log(\bar{\mu}^{(m)}_{01}) \mathbb{1}_{\{x_l > c^*_m\}}\right) \mathbb{1}_{\{l^*_m = l\}}\right\} \\
&=\; \widehat{\lambda}_0(\boldsymbol{x}) \prod_{l=1}^{q} \exp\left\{\sum_{m=1}^{M} \left(\log(\bar{\mu}^{(m)}_{00}) \mathbb{1}_{\{x_l \leq c^*_m\}} + \log(\bar{\mu}^{(m)}_{01}) \mathbb{1}_{\{x_l > c^*_m\}}\right) \mathbb{1}_{\{l^*_m = l\}}\right\}.
\end{aligned}$$

This identity illustrates that we receive a multiplicative structure in the case $J = 2$. This is also nicely illustrated in Figure 7.2. We conclude that for trees of depth 1 we expect the Poisson regression tree boosting machine to have a similar performance as GLMs and GAMs because it (only) allows for multiplicative interactions. For more complex interactions we have to consider more deep SBS regression trees. This finishes this example. ■

Example 7.6 (regression trees, boosting machines and neural networks)**.** In this example we consider regression trees, boosting machines and neural networks on a feature space $\mathcal{X} = [-1, 1]^2$ of dimension $q = q_0 = 2$. For the neural networks we choose the step function activation which makes them directly comparable to regression trees.

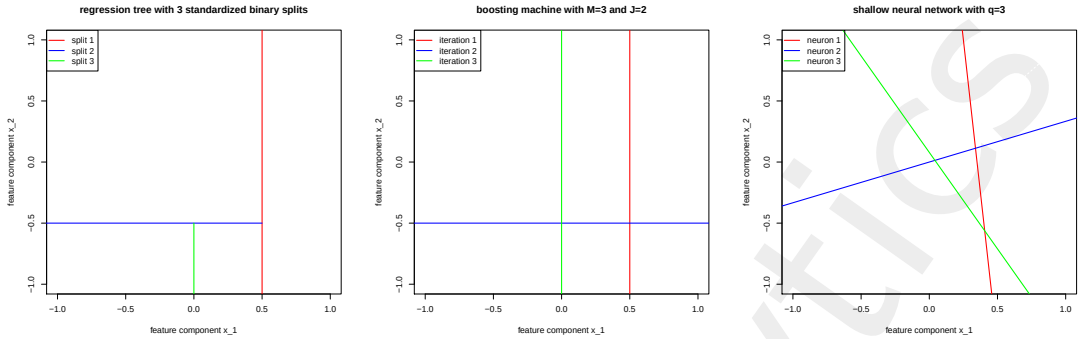


Figure 7.3: (lhs) SBS split regression tree with 3 splits, (middle) Poisson boosting machine with $M = 3$ and $J = 2$, (rhs) shallow neural network with $q_1 = 3$ hidden neurons and step function activation.

We start by comparing the Poisson regression tree boosting machine with one split ($J = 2$) and M iterations to a shallow neural network (of depth $d = 1$) having $q_1 = M$ hidden neurons. These two regression models are illustrated in Figure 7.3 (middle and rhs) for $q_1 = M = 3$. In Example 7.5 we have seen that regression tree boosting machines with $J = 2$ leaves remain in the family of multiplicative models. Shallow neural networks are more general because they allow for splits in any direction, compare Figure 7.3 (middle and rhs). This additional flexibility allows for non-multiplicative interactions and, in fact, the universality theorems on page 105 tell us that these shallow neural networks can approximate a large class of functions if we increase the number of hidden neurons q_1 sufficiently.

Increasing the number of hidden neurons q_1 in a shallow neural network means growing this network in width. On page 116 we have demonstrated that this growing in width may not be very efficient for capturing certain types of interactions, therefore, neural networks should be grown simultaneously in width and depth d to efficiently approximate any type of regression function.

The same can be said for the Poisson regression tree boosting machine. In Figure 7.3 (lhs) we illustrate a SBS regression tree with 3 splits, i.e. having $J = 4$ leaves. Observe that this provides us with a rather complex non-multiplicative interaction in the regression function for the components x_1 and x_2 because the second and third splits do not split across the entire feature space axes, i.e. are non-multiplicative. This complexity is similar to growing a neural network in depth, see (5.16) for an example. The Poisson regression tree boosting machine with $J = 4$ leaves multiplies (for $m \geq 1$) such non-trivial partitions of the type in Figure 7.3 (lhs) which indicates that we may construct very flexible regression functions by iterating the boosting machine for $m \geq 1$ (and for $J > 2$). ■

7.4.3 Example in motor insurance pricing, revisited

The GBM is implemented in the R package `gmb`. In Listing 7.2 we provide a short regression tree boosting algorithm that is based on the R package `rpart`.

Listing 7.2: Poisson regression tree boosting machine

```
1  d   <- 2        # depth of tree
2  M   <- 100      # iterations
3  alpha <- 1     # shrinkage constant
4
5  dat$fit <- dat$expo * sum(dat$claims)/sum(dat$expo)
6
7  for (m in 1:M){
8      tree <- rpart(claims ~ age + ac + power + gas + brand + area + dens + ct
9                            + offset(log(fit)),
10                 data=dat, method="poisson", parms=list(shrink=1),
11                 control=rpart.control(xval=1, minbucket=1000, cp=0.000001,
12                 maxdepth=d, maxsurrogate=0))
13     dat$fit <- predict(tree)^alpha*dat$fit
14                 }
```

Note that the R command `rpart` uses the variable `maxdepth=d` (depth of tree) to determine the size of the tree instead of the number of leaves J. If we use $d = 1$ the tree will have exactly 2 leaves, trees of depth $d = 2$ can have either 3 or 4 leaves, the explicit number will depend on the cost-complexity parameter `cp` on line 11 in Listing 7.2 (and also on the choice of `minbucket`). The variables `xval` and `maxsurrogate` are chosen such that run time is minimized.

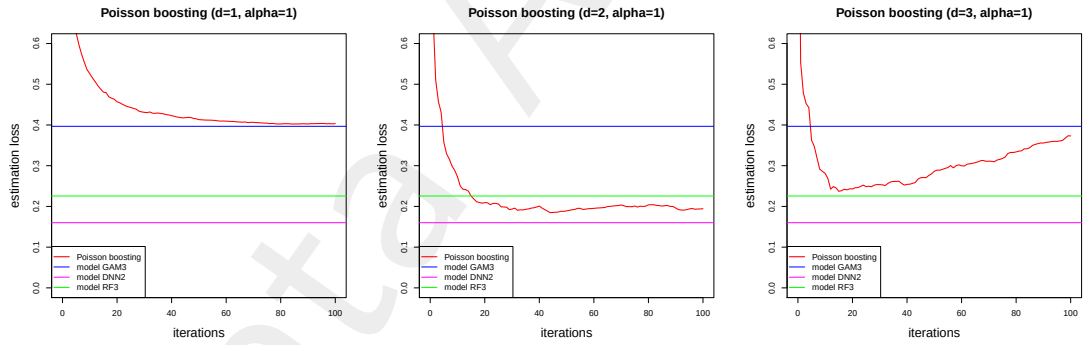


Figure 7.4: Decrease in estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of the Poisson regression tree boosting machine estimators over $M = 100$ iterations of trees of depths $d \in \{1, 2, 3\}$ (lhs, middle, rhs) and shrinkage constant $\alpha = 1$.

We run the Poisson regression tree boosting machine of Listing 7.2 on our MTPL data considered in the previous examples. We choose $M = 100$ iterations, we set the shrinkage constant $\alpha = 1$ and we choose trees of depths $d \in \{1, 2, 3\}$. The resulting estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ are presented in Figure 7.4. The plot on the lhs shows the results for trees of depth $d = 1$. As discussed in Example 7.5, this Poisson regression tree boosting machine can only model multiplicative interactions and, indeed, the corresponding estimation loss converges to the one of the GAM (blue horizontal line) as we increase the number

of iterations M. Thus, this Poisson regression tree boosting machine is not competitive because it does not allow for non-multiplicative interactions.
In Figure 7.4 (middle, rhs) we show the results of the Poisson regression tree boosting machine for trees of depths $d = 2, 3$. We note that these configurations start to be competitive with the neural network and random forest approaches (magenta and green horizontal lines). We also see that the model with trees of depth $d = 3$ starts to over-fit after $M = 20$ iterations.

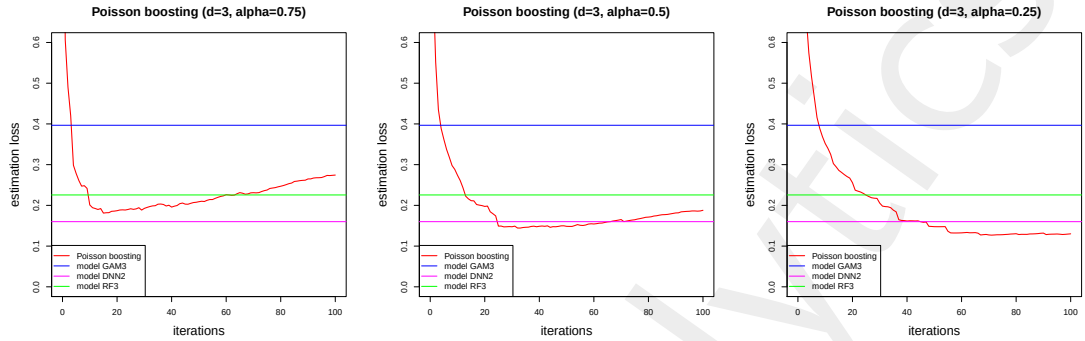


Figure 7.5: Decrease in estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of the Poisson regression tree boosting machine estimators over $M = 100$ iterations of trees of depth $d = 3$ and shrinkage constants $\alpha \in \{0.75, 0.5, 0.25\}$ (lhs, middle, rhs).

In Figure 7.5 we plot the same graphs of the estimation losses $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of the Poisson regression tree boosting machine for trees of depth $d = 3$ and for different shrinkage constants $\alpha \in \{0.75, 0.5, 0.25\}$ (lhs, middle, rhs). For a shrinkage constant of $\alpha = 0.25$, i.e. only a moderate change of the previous estimator in (7.22), we do not receive over-fitting over the first $M = 100$ iterations. Moreover, the resulting boosted regression tree estimator outperforms all regression tree estimators we have met so far. We call this model PBM1, and the corresponding results are reported in Table 7.3.

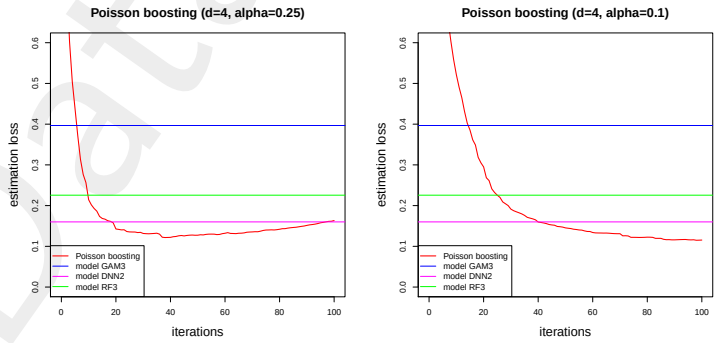


Figure 7.6: Decrease in estimation loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$ of the Poisson regression tree boosting machine estimators over $M = 100$ iterations of trees of depth $d = 4$ and shrinkage constants $\alpha \in \{0.25, 0.1\}$ (lhs, rhs).

Figure 7.6 shows the analogous results of the Poisson regression tree boosting machine

for trees of depth $d = 4$ and $\alpha \in \{0.25, 0.1\}$ (lhs, rhs). This configuration provides an even better performance for shrinkage constant $\alpha = 0.1$, however, at a higher run time. We call this latter model PBM2, and the corresponding results are reported in Table 7.3.

	run time	# param.	CV loss $\mathcal{L}_D^{\text{CV}}$	strat. CV $\mathcal{L}_D^{\text{CV}}$	est. loss $\widehat{\mathcal{E}}(\widehat{\lambda}, \lambda^\star)$	in-sample $\mathcal{L}_D^{\text{is}}$	average frequency
(ChA.1) true model $\lambda^\star$			27.7278				10.1991%
(Ch1.1) homogeneous	0.1s	1	29.1066	29.1065	1.3439	29.1065	10.2691%
(Ch2.4) GLM4	14s	57	28.1502	28.1510	0.4137	28.1282	10.2691%
(Ch3.3) GAM3	50s	79	28.1378	28.1380	0.3967	28.1055	10.2691%
(Ch5.4) CANN1	28s	703	27.9292	27.9362	0.2284	27.8940	10.1577%
(Ch5.5) CANN2	27s	780	27.9306	27.9456	0.2092	27.8684	10.2283%
(Ch5.2) DNN2	123s	703	27.9235	27.9545	0.1600	27.7736	10.4361%
(Ch5.3) DNN3	125s	780	27.9404	27.9232	0.2094	27.7693	9.6908%
(Ch5.7) blended DNN	–	–	–	–	0.1336	27.6939	10.2691%
(Ch6.2) Tree2 $\mathsf{T}^{\min}$	65s	71	–	28.1388	0.3748	27.9156	10.2570%
(Ch7.3) RF3	5'931s	–	27.9808	–	0.2256	27.2942	10.0863%
(Ch7.4) PBM1	99s	725	27.8794	27.8783	0.1301	27.6149	10.2655%
(Ch7.5) PBM2	115s	1314	27.8686	27.8763	0.1254	27.5921	10.2588%

Table 7.3: Poisson deviance losses of K-fold cross-validation (1.11) with $K = 10$, corresponding estimation loss (1.13), and in-sample losses (1.10); green color indicates values which can only be calculated because we know the true model $\lambda^\star$; losses are reported in 10^{-2}; run time gives the time needed for model calibration, and '# param.' gives the number of estimated model parameters, this table follows up from Table 7.2.

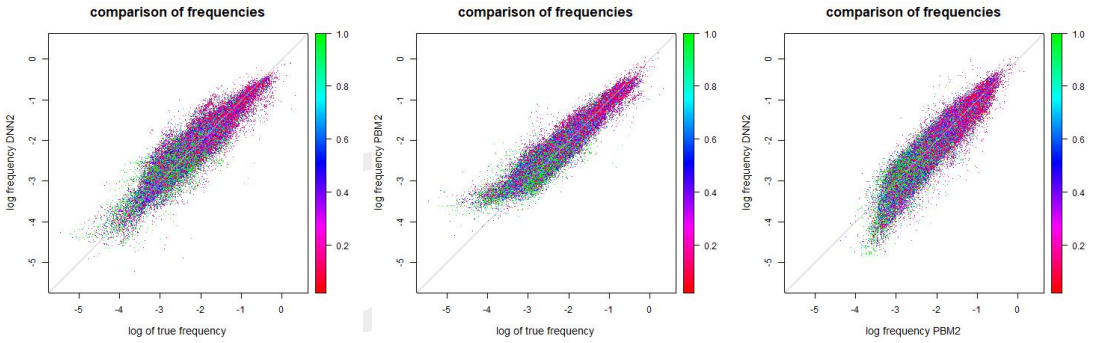


Figure 7.7: Resulting estimated frequencies (on log scale) of true vs. model DNN2 (lhs), true vs. model PBM2 (middle), and model PBM2 vs. DNN2 (rhs).

From these results we conclude that the boosting machine provides excellent results, on the same level as neural network ensembles, see Table 7.3, but at a reasonable run time. Another advantage of the Poisson regression tree boosting machine is that it provides rather stable average frequency estimates (last column of Table 7.3). In fact, the balance property only fails to hold here because we use the Bayesian version of the parameter estimates. Neural networks are also competitive in terms of estimation loss and run time, but the average frequency estimates need balance property regularization as shown in Section 5.1.5. Neural networks have the advantage that they allow for continuous inter-

and extrapolation if the activation functions have been chosen to be continuous.
In Figure 7.7 we compare the log-frequencies of the neural network model DNN2, the Poisson regression tree boosting machine PBM2 and the true model $\lambda^\star$. We see that the Poisson regression tree boosting machine is more concentrated around the true model than the neural network model, compare Figure 7.7 (lhs) and (middle). The only concern that we see with the Poisson regression tree boosting machine is that the low frequencies in the left-lower corner of Figure 7.7 (middle) are biased. This may be caused by the fact that we set a minimal leaf size of 1'000 cases, see line 11 of Listing 7.2. This leaf size may imply that we cannot distinguish the quality of good insurance policies. We could decrease this value. However, this comes at the price of more run time and with a higher potential of over-fitting. This closes our example.

7.4.4 AdaBoost algorithm

Usually, boosting is explained with the AdaBoost algorithm at hand which goes back to Freund–Schapire [45]. In analogy to the Poisson regression tree boosting machine, we can design a boosting algorithm for binary classification, see Section 2.5.1. We assume to have a binary classification problem

$$\mathcal{C} : \mathcal{X} \to \mathcal{Y} = \{0, 1\}, \qquad \boldsymbol{x} \mapsto y = \mathcal{C}(\boldsymbol{x}),$$

which needs to be estimated from the data

$$\mathcal{D} = \{(Y_1, \boldsymbol{x}_1), \ldots, (Y_n, \boldsymbol{x}_n)\}, \tag{7.23}$$

with features $\boldsymbol{x}_i \in \mathcal{X}$ and binary responses $Y_i \in \{0, 1\}$. For a given classifier $\mathcal{C}$ we can study the misclassification rate on the data $\mathcal{D}$ given by, we also refer to (2.31)-(2.32) and (6.30),

$$\mathcal{L}_{\mathbb{1}_{\{\neq\}}} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{Y_i \neq \mathcal{C}(\boldsymbol{x}_i)\}}.$$

A *weak classifier* $\mathcal{C}$ is one that is (only slightly) better than random guessing, a more precise definition is given in (7.28), below. Boosting constructs a sequence $(\widehat{\mathcal{C}}_m)_{m=1,\ldots,M}$ of weak classifiers together with a sequence of weights $(\alpha_m)_{m=1,\ldots,M}$ from which a more powerful classifier is constructed. The Ada(ptive)Boost algorithm works as follows.

AdaBoost Algorithm.

---

(0) Initialize working weights $w_i^{(1)} = 1$ for $i = 1, \ldots, n$.

(1) Repeat for $m = 1, \ldots, M$

(a) fit a weak classifier $\widehat{\mathcal{C}}_m$ to the working data

$$\mathcal{D}^{(m)} = \left\{ \left(Y_1, \boldsymbol{x}_1, w_1^{(m)}\right), \ldots, \left(Y_n, \boldsymbol{x}_n, w_n^{(m)}\right) \right\}, \tag{7.24}$$

where we attach the working weights $(w_i^{(m)})_{i=1,\ldots,n}$ to the original data (7.23);

(b) calculate the *weighted (in-sample) misclassification rate* of $\widehat{\mathcal{C}}_m$ on $\mathcal{D}^{(m)}$ given by

$$\mathcal{L}^{(m)}_{w\mathbb{1}_{\{\neq\}}} = \frac{1}{\sum_{i=1}^n w_i^{(m)}} \sum_{i=1}^n w_i^{(m)} \mathbb{1}_{\{Y_i \neq \widehat{\mathcal{C}}_m(\boldsymbol{x}_i)\}};$$

the algorithm is terminated if $\mathcal{L}^{(m)}_{w\mathbb{1}_{\{\neq\}}} = 0$, otherwise

(c) compute

$$\alpha_m = \log\left(\frac{1 - \mathcal{L}^{(m)}_{w\mathbb{1}_{\{\neq\}}}}{\mathcal{L}^{(m)}_{w\mathbb{1}_{\{\neq\}}}}\right);$$

(d) update the working weights $i = 1, \ldots, n$

$$w_i^{(m+1)} = w_i^{(m)} \exp\left\{\alpha_m \mathbb{1}_{\{Y_i \neq \widehat{\mathcal{C}}_m(\boldsymbol{x}_i)\}}\right\}. \tag{7.25}$$

(2) Return the classifier (weighted majority vote)

$$\boldsymbol{x} \quad \mapsto \quad \widehat{\mathcal{C}}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{m=1}^M \alpha_m \left(\widehat{\mathcal{C}}_m(\boldsymbol{x}) - 0.5\right)\right). \tag{7.26}$$

Comments on the AdaBoost algorithm.

- In (7.24) the observations $\mathcal{D}$ are extended by the working weights $(w_i^{(m)})_{i=1,\ldots,n}$. These working weights may be interpreted to give more attention to particular misclassification than to others, i.e. we pay more attention to cases $(Y_i, \boldsymbol{x}_i)$ with a higher working weight $w_i^{(m)}$. Such weights have not been used in classification, yet, for instance, in the binary tree classification algorithm of Section 6.3, but they could be implemented by modifying the empirical probabilities in (6.26) accordingly to

$$\widehat{p}(y, \mathcal{X}'; \mathcal{D}^{(m)}) = \frac{\sum_{i=1}^n w_i^{(m)} \mathbb{1}_{\{Y_i = y,\ \boldsymbol{x}_i \in \mathcal{X}'\}}}{\sum_{i=1}^n w_i^{(m)}}. \tag{7.27}$$

Observe that we initialize the above algorithm by setting $w^{(1)} \equiv 1$ which provides

$$\widehat{p}(y, \mathcal{X}'; \mathcal{D}^{(1)}) = \frac{\sum_{i=1}^n \mathbb{1}_{\{Y_i = y,\ \boldsymbol{x}_i \in \mathcal{X}'\}}}{n} = \frac{n(y, \mathcal{X}'; \mathcal{D})}{\sum_{y \in \mathcal{Y}} n(y, \mathcal{X}; \mathcal{D})}.$$

Thus, a first classifier $\widehat{\mathcal{C}}_1$ for $m = 1$ in the AdaBoost algorithm can be constructed by the (classical) classification tree algorithm presented in Section 6.3.

- Since $\widehat{\mathcal{C}}_m$ is a weak classifier for the working weights $(w_i^{(m)})_{i=1,\ldots,n}$, it is slightly better than random guessing for these weights, which is defined by the corresponding weighted (in-sample) misclassification rate satisfying requirement

$$\mathcal{L}^{(m)}_{w\mathbb{1}_{\{\neq\}}} < 1/2. \tag{7.28}$$

This implies that $\alpha_m > 0$ and henceforth update (7.25) increases the working weights $w_i^{(m+1)}$ of the cases $(Y_i, \boldsymbol{x}_i)$ that are misclassified by $\widehat{\mathcal{C}}_m$, thus, more attention is paid to these misclassified cases in the next (weak) classifier $\widehat{\mathcal{C}}_{m+1}$.

- In this sense, classifier (7.26) can be viewed as a GAM with basis functions $\widehat{\mathcal{C}}_1(\cdot), \ldots, \widehat{\mathcal{C}}_M(\cdot)$, see (3.10). The main differences are that these basis functions are constructed on site (*adaptive*) and the optimization is *stage-wise* because only the latest parameter α_m is optimized (and the previous ones $\alpha_1, \ldots, \alpha_{m-1}$ are kept fixed).
- It can be shown that the AdaBoost algorithm arises similarly to the Poisson regression tree boosting algorithm as a stage-wise adaptive optimization algorithm from an in-sample loss optimization problem. In this case we consider a classification problem with either having the exponential function or the binomial deviance loss as objective function, this also explains the choices of α_m, for details we refer to Hastie et al. [62], Sections 10.4-10.5.

Chapter 8

Telematics Car Driving Data

In the previous chapters we have been analyzing claims frequency modeling of a heterogeneous car insurance portfolio. The risk drivers of this heterogeneous portfolio have been described by classical car insurance feature information. This classical car insurance feature information can be divided into several different groups:

- **driver specific features:** age of driver, gender of driver, marital status, date and place of driving test, occupation, medical conditions, size of household, type of flat, garage, credit record, leasing, etc.
- **car specific features:** type of car, car brand, size of car, weight of car, age of car, horse power, type of engine, cubic capacity, price of car, equipment, number of seats, etc.
- **insurance contract specific features:** type of contract, duration of contract, issue date of contract, sales channel, deductible, other insurance covers, etc.
- **geographic features:** province of living, zip code, city-rural area, etc.
- **driving related features:** annual distance, vehicle use, bonus-malus level, claims experience, etc.

Typical car insurance tariffs are based on more than 30 feature components. From the previous list, however, we see that many of these components are not directly related to the driving habits, driving styles and driving skills of the drivers.
Nowadays, many vehicles are equipped with technology that transmits car driving information via telecommunication systems to central data warehouses. These data transmissions (called telematics data) comprise detailed car driving information, and in the foreseeable future this information will be used for car insurance pricing because it displays driving habits and driving styles rather transparently. In fact, it is likely that this information will complement the classical feature information from above.
These car driving transmissions may comprise rather different information (depending on the installed devices). This information may include high-frequency data about location (typically GPS location sec by sec), speed, acceleration and deceleration, left- and right-turns, engine revolutions (all sec by sec). Moreover, it may include number of trips, total distance of trips, total duration of trips, time stamp of trips, road conditions, road types,

weather information, as well as driver related information. The main difficulty from a statistical point of view is to convert this high-dimensional and high-frequency data into useful feature information for car insurance pricing.

If we assume that in average a car driver produces 100 KB of telematics data per day, this amounts to 40 MB of telematics data per year. For a comparably small portfolio of 100'000 car drivers this results in 4 TB of telematics data each year. This is a large amount of data, and proper data warehousing is crucial to handle this data. For instance, it may already be difficult to identify a given driver each day in that data, thus, even calculating the total annual distance of a given driver may result in a non-trivial exercise. From this it is obvious that we have to compress telematics data in an appropriate way to make it useful for statistical analysis.

In the actuarial literature there is an increasing number of contributions that study telematics data from a statistical point of view. Verbelen et al. [129] use GAMs to analyze the effect of telematics data in conjunction with classical feature information. Their study considers information like number of trips, total distance driven, road type, and daytime and weekday of driving. It does not study speed, acceleration and deceleration, intensity of left- and right-turns. Another interesting stream of literature explores pay-as-you-drive (PAYD) and usage-based (UB) car insurance products, see Ayuso et al. [5, 6] and Boucher et al. [12]. These works elaborate on the exposure measures and design pricing frameworks that are directly related to the effectively driven distances. The latter work is complemented by Denuit et al. [30] which adds to the above features additional information about distances traveled at night, distances driven in urban zones, and distances driven above speed limits. This information is still rather *driving habits* based, except the last one on 'excesses of speed limits' which indicates information about *driving styles*.

Probably, the first contribution in the actuarial literature that considers driving styles from telematics car driving data is Weidner et al. [132]. These authors use Fourier analysis for pattern recognition to study driving behavior and driving styles. In particular, they analyze the frequency spectrum obtained by single driving maneuvers and trips. Though, the main question of how to use these maneuvers and trip information as feature information for car insurance pricing is not finally answered in Weidner et al. [132], yet. In this chapter, we start by considering Wüthrich [136] who classifies telematics information of different car drivers into different groups of drivers according to a chosen similarity measure. This classification does not use claims frequency information and belongs to the field of *unsupervised learning methods*, dealing with cluster analysis and pattern recognition. Based on this starting point we will discuss other unsupervised learning methods for clustering different drivers.

8.1 Description of telematics car driving data

8.1.1 Simple empirical statistics

For the analysis in these notes we consider telematics car driving data which comprises GPS location data sec by sec. In Figure 8.1 we choose three different car drivers (call them drivers A, B and C, respectively) and we illustrate 200 individual trips of these

three car drivers. For confidentiality reasons and illustrative purposes all individual trips are initialized to start at location $(0, 0)$, and they are randomly rotated (at this origin) and potentially reflected at the coordinate axes. These transformations do not change the crucial driving characteristics like speed, acceleration and deceleration, and intensity of turns.

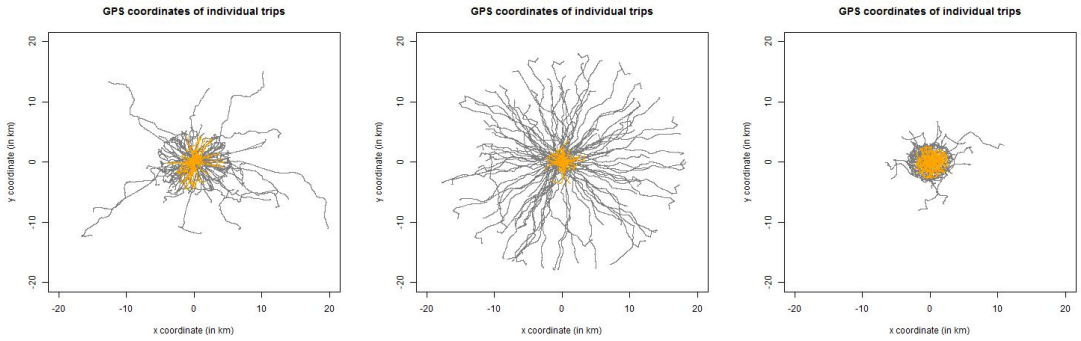


Figure 8.1: 200 individual trips of the three different car drivers A (left), B (middle) and C (right); in orange color are the shorter 100 trips and in gray color the longer 100 trips; the square shows the area $[-20 \text{ km}, 20 \text{ km}]^2$.

Figure 8.1 shows for each of these three drivers the square $[-20 \text{ km}, 20 \text{ km}]^2$, in orange color we plot the shorter 100 trips and in gray color the longer 100 trips. We see that driver A usually travels short distances (within a radius of less than 5 km) but he also drives a few longer trips; driver B is a long-distance traveler (with many trips longer than 10 km); and driver C only does shorter drives.
From this GPS location data we can calculate many other quantities. Following [51], we calculate the average speed v_t, the average acceleration and deceleration a_t, and the average change in direction (angle) Δ_t every second t of each individual trip. Let (x_t, y_t) denote the GPS location in meters every second t of a single trip of a given driver. From this GPS location data we calculate the average speed (velocity) at time t (in m/s)

$$v_t = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2},$$

and the average acceleration and deceleration at time t (in m/s^2)

$$a_t = v_t - v_{t-1}.$$

For a positive average speed $v_t > 0$ at time t, we define the direction of the heading by

$$\varphi_t = \operatorname{atan2}(y_t - y_{t-1}, x_t - x_{t-1}) \quad \in (-\pi, \pi],$$

where atan2 is a common modification of the arctan function that transforms Cartesian coordinates to polar coordinates such that the resulting polar angle is in $(-\pi, \pi]$. For positive speeds $v_s > 0$ at times $s = t-1, t$ we can then consider the change in direction (angle) from $t-1$ to t given by $\varphi_t - \varphi_{t-1} \in (-2\pi, 2\pi)$. For the change in direction (angle) at time t we define

$$\Delta_t = |\sin(\varphi_t - \varphi_{t-1})|.$$

We choose absolute values of changes in angles because our telematics data analysis should not be influenced by the signs of the angles, but rather by the intensity of turns (no matter whether these are left- or right-turns). Moreover, we choose the sine of the change in angle: the reason for this choice is that GPS location data has some imprecision[1] which manifests stronger at very low speeds, say, when the car is almost standing still. Taking the sine of the change in angle slightly dampens this effect, but requires $\varphi_t - \varphi_{t-1} \in (-2\pi, -3\pi/2] \cup [-\pi/2, \pi/2] \cup [3\pi/2, 2\pi)$ for identifiability reasons. Note that the latter is usually fulfilled because changes of directions within one second cannot exceed $\pi/2$.

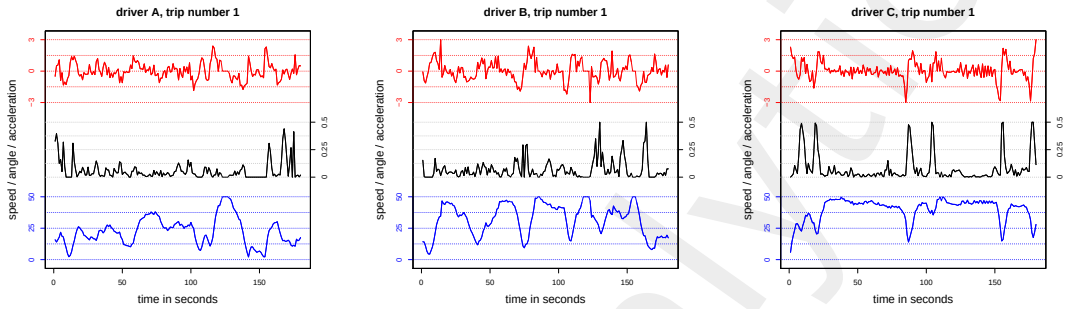


Figure 8.2: Individual trips of the three drivers A (left), B (middle) and C (right): the lower line in blue color shows the speeds $(v_t)_t$ (in km/h), the upper line in red color shows that acceleration and deceleration $(a_t)_t$ (in m/s^2), and the middle line in black color shows the changes in angle $(\Delta_t)_t$ over 180 sec, i.e. $t \in \{0, \ldots, 180\}$.

In Figure 8.2 we illustrate individual trips of lengths 180 sec of the three drivers A (left), B (middle) and C (right). The lower lines in blue color show the speed (velocity) patterns $(v_t)_t$ (in km/h), the upper lines in red color show that acceleration and deceleration patterns $(a_t)_t$ (in m/s^2), and the middle lines in black color show the changes in angle patterns $(\Delta_t)_t$. We note that changes in angle are bigger at lower speeds. From Figure 8.2 (rhs) we can very well see that these changes in angle often go along with deceleration first and then accelerating after changing the direction of the heading.

We remark that the maximal acceleration and deceleration a_t has been capped at ± 3m/s^2 for the plots in Figure 8.2. [133] state that normal acceleration goes up to 2.5m/s^2, and extreme acceleration can go up to 6m/s^2 for vehicles driving straight ahead. Braking

[1] GPS location data is typically subject to quite some imprecision. First of all, often GPS location data is rounded. This provides a first source of imprecision which has a stronger influence on acceleration and changes in angle at speeds close to zero. A second source of imprecision may be that the GPS signal itself is not fully precise w.r.t. position and timing. Finally, it may happen that the GPS signal is not received at all, for instance, while driving through a tunnel. The latter can be identified more easily because it leads to missing values or accelerations beyond physical laws (if missing values are not marked). In many cases, one also directly receives speed and acceleration (in all directions) from installed devices. However, also this data is subject to imprecision. In particular, often one faces the issue that the devices are not correctly calibrated, for instance, there may be a transmission of a constant positive speed of a car standing still according to GPS location data. Moreover, depending on the type of device installed the speed may be rounded to km/h which may be too rough, etc., and other devices may calculate statistics on a coarser time scale due to data volume constraints.

may be stronger and may vary up to -8m/s^2. In our data acceleration and deceleration beyond $\pm 3\text{m/s}^2$ is rather sparse and may also be caused by data quality reasons of an imprecise GPS signal, therefore we typically cap extreme acceleration and deceleration.

	driver A	driver B	driver C
total distance (in km)	1'235	1'808	1'001
average distance per trip (in km)	6.18	9.04	5.01
total time (in h)	40.84	51.27	39.43
average time per trip (in min)	12.15	15.38	11.83
average velocity (in km/h)	30.25	35.27	25.39
median velocity over trips (in km/h)	28.83	35.91	25.29

Table 8.1: Empirical statistics of the drivers A, B and C.

In Table 8.1 we provide some simple empirical statistics of these three selected drivers, these include the totally driven distance of all considered trips and the total time therefore used. We also provide the average trip lengths (in distance and time), the average velocity over all trips, and the median speed of the single trips is provided. These statistics reflect the graphs in Figure 8.1. Next we analyze the amount of time spent in different speed buckets. We therefore consider the speed intervals (in km/h)

$$\begin{array}{ll} [0] & \text{car is in idling mode (zero speed),} \\ (0,5] & \text{acceleration or braking phase (from/to speed 0),} \\ (5,20] & \text{low speeds,} \\ (20,50] & \text{urban area speeds,} \\ (50,80] & \text{rural area speeds,} \\ (80,130] & \text{highway speeds (we truncate speeds above 130 km/h).} \end{array} \tag{8.1}$$

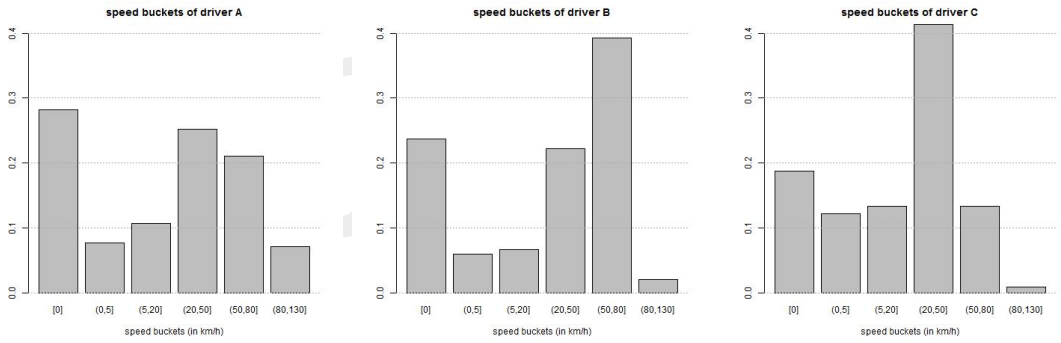


Figure 8.3: Speed bucket distribution (in driving time) of the three drivers A, B and C.

In Figure 8.3 we show the amount of time spent in each of these speed buckets for the three selected drivers A, B and C. We observe that the distribution of the resulting speeds varies considerably over the speed buckets; not surprisingly driver B drives almost half of his time with speeds above 50 km/h, whereas the other two drivers mostly drive with speeds below 50 km/h. Also remarkable is that driver A stands still 28% of his total trip time of 40.84 hours, the corresponding figures for drivers B and C are 24% of 51.27 hours and 19% of 39.43 hours, respectively. These numbers are rather typical for big

cities, for instance, the idling mode in peak hours on arterials in the city of Beijing takes roughly 29% of the total driving time and in the city of Shanghai 38% of the total driving time, see Table 6 in Wang et al. [131]. Of course, these numbers depend on the network topography of a city and therefore may vary considerably.
Up to now we have only been considering driving habits, i.e. whether we have a long-distance driver, an urban area driver, etc. We could calculate many more of these simple empirical statistics. These statistics can be transformed into feature information which provides important complementary information to the classical actuarial feature information. However, the statistics considered so far do not say much about driving styles. This analysis is our next aim.

8.1.2 The velocity-acceleration heatmap

To analyze driving styles we consider so-called velocity-acceleration (v-a) heatmaps that display the average velocity v on the x-axis (in km/h) and the corresponding acceleration and deceleration a on the y-axis (in m/s^2). For this analysis we consider the same speed buckets for the velocity as used in Figure 8.3. Speed buckets have the advantage that the different driving habits do not directly influence the v-a heatmap analysis, if we consider in each speed bucket the resulting empirical density (normalized to 1). To receive the v-a heatmap, say in $R = (5, 20] \times [-2, 2]$ (in km/h$\times$m/s^2), we partition R into J congruent rectangles $R_1, \ldots, R_J$ with

$$\bigcup_{j=1}^{J} R_j = R \qquad \text{and} \qquad R_j \cap R_{j'} = \emptyset \quad \text{for all } j \neq j'. \tag{8.2}$$

Let $x_j \geq 0$ denote the total relative time amount spent in R_j of a given driver. Then, $\boldsymbol{x} = (x_1, \ldots, x_J)' \in \mathcal{P}_J$ gives us a discrete probability distribution in the $(J-1)$-unit simplex in $\mathbb{R}_+^J$ satisfying

$$x_j \geq 0 \ \text{ for all } j = 1, \ldots, J, \text{ and } \ \sum_{j=1}^{J} x_j = 1.$$

The corresponding v-a heatmap is defined to be the graphical illustration of $\boldsymbol{x}$ on R. In Figure 8.4 we provide these graphical illustrations for our three selected drivers on the different speed buckets defined in (8.1): the column on the left-hand side shows driver A, the middle column gives driver B and the column on the right-hand side corresponds to driver C. The five different rows show the five non-zero speed buckets. In the first row we have the acceleration and deceleration styles of the three drivers in speed bucket $(0, 5]$ km/h. We observe that these look quite differently. Driver B seems to accelerate more intensively than driver C at speed 0, and it seems that he brakes more heavily at a higher speed than driver C. This may indicate that driver B is a more aggressive driver than driver C (because he needs to brake more abruptly, i.e. he approaches a red light at a higher speed). Driver A has intermediate braking and acceleration maxima, this indicates that he drives a manual gear car. The same consideration applies to the speed bucket $(5, 20]$ km/h in the second row of Figure 8.4, in particular, the v-a heatmaps of drivers B and C look much more smooth which is probably implied by driving an automatic gear car (compared to driver A).

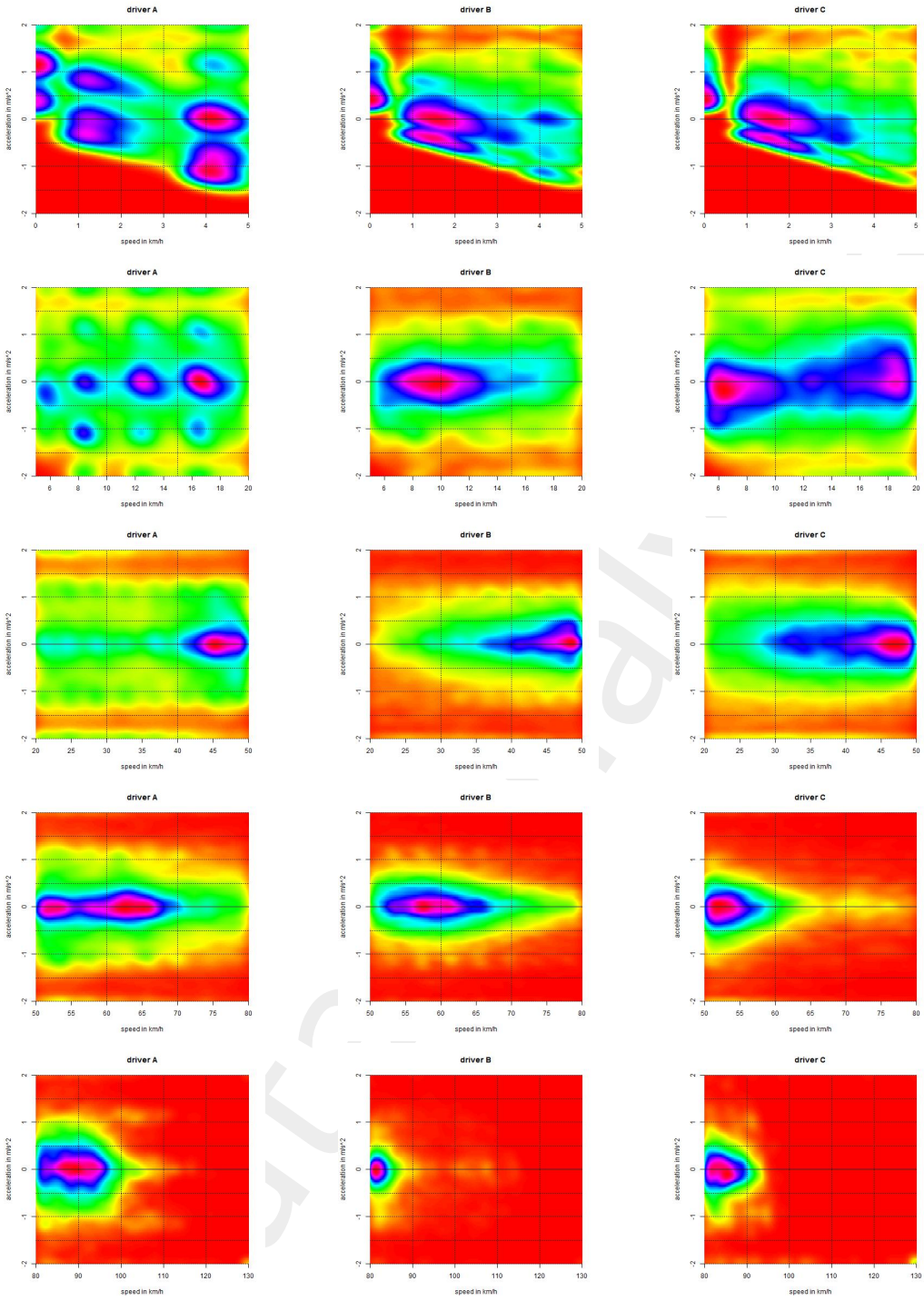


Figure 8.4: v-a heatmaps of the three selected car drivers A, B and C in the different speed buckets $(0, 5]$, $(5, 20]$, $(20, 50]$, $(50, 80]$ and $(80, 130]$ km/h, respectively.

Rows three and four of Figure 8.4 are interpreted such that driver B has the smoothest driving style in these two speed buckets because the vertical diameter (level sets) of his heatmaps are smaller compared to the other two drivers. This may also be caused by

the possibility that driver B drives a heavier car that is less agile than the other two drivers. Note also that the light upper and lower acceleration boundaries in these graphs are caused by the fact that we truncate (here) acceleration and deceleration at ± 2 m/s^2. Finally, the last row of Figure 8.4 shows the driving styles above speed 80 km/h. We observe that in this speed bucket we only have limited information because our three drivers only spend little time in this speed bucket. Similar graphs could be provided for left- and right-turns.

Remarks.

- There is a closely related stream of literature that aims at understanding vehicular emissions, energy consumption and impacts on traffic, see [40, 66, 70, 77, 131]. This literature aims at constructing typical driving cycles for cities considering the respective topology of the given city. For the selection of representative driving cycles this literature uses so-called speed acceleration probability distributions (SAPD), see Figures 5-6 in [70], and speed acceleration matrices, see [77], which are nothing else than our v-a heatmaps.

- In Gao et al. [49] we have been studying the robustness of v-a heatmaps. Convergence results show that roughly 300 minutes of driving experience are sufficient to receive stable v-a heatmaps in the speed bucket $(5, 20]$ km/h.

8.2 Cluster analysis

Our main goal is to classify the v-a heatmaps, i.e. we would like to identify the drivers that have similar v-a heatmaps. These drivers are then interpreted to have similar driving styles (according to their v-a heatmaps), and are therefore put into the same categorical classes for car insurance pricing. Here, we do this for one single speed bucket only. We choose $R = (5, 20] \times [-2, 2]$ (in km/h$\times$m/s^2) because this speed bucket has been identified to be very predictive for claims frequency modeling, see Gao et al. [52]. We partition this rectangle R as described in (8.2), and we denote the resulting discrete probability distributions in the $(J-1)$-unit simplex by $\mathcal{P}_J$. Classification of individual car drivers is then achieved by looking at a classifier function

$$\mathcal{C} : \mathcal{P}_J \to \mathcal{K}, \qquad \boldsymbol{x} \mapsto \mathcal{C}(\boldsymbol{x}), \tag{8.3}$$

with $\mathcal{K} = \{1, \ldots, K\}$ describing the K different categorical classes considered. Thus, classification aims at finding such a classifier $\mathcal{C}$ that separates different driving styles. We will present a simple unsupervised machine learning method for this task. For more sophisticated methods on pattern recognition we refer to the related literature.

8.2.1 Dissimilarity function

In order to construct a classifier $\mathcal{C}$ we start by describing the dissimilarity between two different probability distributions $\boldsymbol{x}_i, \boldsymbol{x}_l \in \mathcal{P}_J$. The dissimilarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_l$ is

defined by

$$d(\boldsymbol{x}_i, \boldsymbol{x}_l) \;=\; d_w(\boldsymbol{x}_i, \boldsymbol{x}_l) \;=\; \frac{1}{2} \sum_{j=1}^{J} w_j \left(x_{i,j} - x_{l,j}\right)^2,$$

where $w_j \geq 0$ are predefined weights. Remark that we choose the (weighted) square distance because it has nice analytical properties (as we will see below), other choices are described in Section 14.3.2 of Hastie et al. [62]. The weights $w = (w_j)_j$ may allow us to emphasize dissimilarities on certain subsets R_j more than on others. Below, for simplicity, we only consider $w \equiv 1$.

Assume we have n different car drivers with v-a heatmaps $\boldsymbol{x}_i \in \mathcal{P}_J$ for $i = 1, \ldots, n$. The *total dissimilarity* over all drivers is defined by

$$D(\mathcal{I}) \;=\; \frac{1}{n} \sum_{i,l=1}^{n} d(\boldsymbol{x}_i, \boldsymbol{x}_l).$$

Lemma 8.1. *The total dissimilarity over all drivers satisfies*

$$D(\mathcal{I}) \;=\; \sum_{j=1}^{J} w_j \sum_{i=1}^{n} \left(x_{i,j} - \bar{x}_j\right)^2,$$

with (empirical) means $\bar{x}_j = n^{-1} \sum_{i=1}^{n} x_{i,j}$*, for* $j = 1, \ldots, J$*.*

Proof of Lemma 8.1. A straightforward calculation provides

$$\begin{aligned} D(\mathcal{I}) \;&=\; \frac{1}{2n} \sum_{i,l=1}^{n} \sum_{j=1}^{J} w_j \left(x_{i,j} - x_{l,j}\right)^2 \;=\; \frac{1}{2n} \sum_{j=1}^{J} w_j \sum_{i,l=1}^{n} \left(x_{i,j} - \bar{x}_j + \bar{x}_j - x_{l,j}\right)^2 \\ &=\; \frac{1}{2n} \sum_{j=1}^{J} w_j \sum_{i,l=1}^{n} \left(x_{i,j} - \bar{x}_j\right)^2 + 2\left(x_{i,j} - \bar{x}_j\right)\left(\bar{x}_j - x_{l,j}\right) + \left(\bar{x}_j - x_{l,j}\right)^2 \\ &=\; \sum_{j=1}^{J} w_j \sum_{i=1}^{n} \left(x_{i,j} - \bar{x}_j\right)^2 + \frac{1}{n} \sum_{j=1}^{J} w_j \sum_{i,l=1}^{n} \left(x_{i,j} - \bar{x}_j\right)\left(\bar{x}_j - x_{l,j}\right). \end{aligned}$$

The second term is zero and the claim follows. □

Lemma 8.1 has a nice interpretation: $\bar{x}_j$ is the empirical mean of all probability weights $x_{1,j}, \ldots, x_{n,j}$ of the n drivers on subset R_j and the dissimilarity in these probability weights is measured by the (empirical) variance defined by

$$s_j^2 \;=\; \frac{1}{n} \sum_{i=1}^{n} \left(x_{i,j} - \bar{x}_j\right)^2.$$

Thus, the total dissimilarity over all drivers is given by

$$D(\mathcal{I}) \;=\; n \sum_{j=1}^{J} w_j \; s_j^2.$$

The weights $w_j \geq 0$ can now be chosen such that different subsets R_j are weighted differently. For instance, if we are concerned with high acceleration, then we would give more weight to subsets R_j that cover the high acceleration region in R. In the examples below we will choose $w \equiv 1$. The following lemma is immediate.

Lemma 8.2. *The empirical means $\bar{x}_j$ are optimal in the sense that*

$$\bar{x}_j \;=\; \operatorname*{argmin}_{m_j} \sum_{i=1}^{n} (x_{i,j} - m_j)^2 .$$

Lemma 8.2 says that the total dissimilarity $D(\mathcal{I})$ is found by solving an optimization problem on each subset R_j. Moreover, this optimal solution $(\bar{x}_j)_{j=1,\ldots,J}$ is itself a probability distribution on R because it can be interpreted as a mixing distribution: all elements satisfy $\bar{x}_j \in [0,1]$ and

$$\sum_{j=1}^{J} \bar{x}_j \;=\; \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{J} x_{i,j} \;=\; 1.$$

Thus, $(\bar{x}_j)_{j=1,\ldots,J} \in \mathcal{P}_J$ is the discrete probability distribution that solves the optimization problems in Lemma 8.2 simultaneously for all $j = 1, \ldots, J$.

8.2.2 Classifier and clustering

In the previous section we have considered the total dissimilarity over all drivers given by

$$D(\mathcal{I}) \;=\; \frac{1}{2n} \sum_{i,l=1}^{n} \sum_{j=1}^{J} w_j \, (x_{i,j} - x_{l,j})^2 \;=\; \sum_{j=1}^{J} w_j \sum_{i=1}^{n} (x_{i,j} - \bar{x}_j)^2 \;=\; n \, \sum_{j=1}^{J} w_j \; s_j^2.$$

This corresponds to a weighted sum of squares (WSS), additionally scaled by n^{-1}. This WSS is an error measure if we do not assume any additional model structure, i.e. under the assumption that we have a homogeneous portfolio of drivers. In the subsequent analysis we introduce model structure, that aims at separating different drivers into homogeneous sub-classes. We introduce a classification structure by partitioning the set $\mathcal{I} = \{1, \ldots, n\}$ of all drivers into K (non-empty) *clusters* $\mathcal{I}_1, \ldots, \mathcal{I}_K$ satisfying

$$\bigcup_{k=1}^{K} \mathcal{I}_k = \mathcal{I} \qquad \text{and} \qquad \mathcal{I}_k \cap \mathcal{I}_{k'} = \emptyset \quad \text{for all } k \neq k'.$$

These K clusters define a natural classifier $\mathcal{C}$, restricted to the drivers $i \in \mathcal{I}$, given by

$$\mathcal{C} : \mathcal{I} \to \mathcal{K}, \qquad i \mapsto \mathcal{C}(i) = \sum_{k=1}^{K} k \; \mathbb{1}_{\{i \in \mathcal{I}_k\}}.$$

Note that we use a slight abuse of notation here: at the moment the classifier $\mathcal{C}$ is only defined on $\mathcal{I}$, this is in contrast to (8.3). Its extension to $\mathcal{P}_J$ is provided in Remarks 8.4, below.

Under this classification approach on $\mathcal{I}$ we can decompose the WSS into two parts, the within-cluster sum of squares (WCSS) and the between-cluster sum of squares (BCSS), the latter being explained by the chosen clustering structure. That is,

$$\begin{aligned} D(\mathcal{I}) \;&=\; \sum_{k=1}^{K} \sum_{j=1}^{J} w_j \sum_{i \in \mathcal{I}_k} (x_{i,j} - \bar{x}_j)^2 \\ &=\; \sum_{k=1}^{K} \sum_{j=1}^{J} w_j \sum_{i \in \mathcal{I}_k} \left(x_{i,j} - \bar{x}_{j|k}\right)^2 \;+\; \sum_{k=1}^{K} n_k \sum_{j=1}^{J} w_j \left(\bar{x}_{j|k} - \bar{x}_j\right)^2, \end{aligned} \tag{8.4}$$

where $n_k = |\mathcal{I}_k|$ is the number of drivers in $\mathcal{I}_k$ and the empirical means on $\mathcal{I}_k$ are given by

$$\bar{x}_{j|k} = \frac{1}{n_k} \sum_{i \in \mathcal{I}_k} x_{i,j}. \tag{8.5}$$

The last term on the right-hand side of (8.4) is interpreted as the *between-cluster dissimilarity* of classifier $\mathcal{C}$, defined by

$$B(\mathcal{C}) = \sum_{k=1}^{K} n_k \sum_{j=1}^{J} w_j \left(\bar{x}_{j|k} - \bar{x}_j\right)^2 \ \geq \ 0.$$

This is term that can be explained by the classification structure induced by the partition $(\mathcal{I}_k)_k$ of $\mathcal{I}$. The first term on the right-hand side of (8.4) is the *aggregate within-cluster dissimilarity* of classifier $\mathcal{C}$, defined by

$$W(\mathcal{C}) \ = \ \sum_{k=1}^{K} \sum_{j=1}^{J} w_j \sum_{i \in \mathcal{I}_k} \left(x_{i,j} - \bar{x}_{j|k}\right)^2. \tag{8.6}$$

Thus, we immediately get the following simple corollary.

Corollary 8.3. *For any partition $(\mathcal{I}_k)_{k \in \mathcal{K}}$ of $\mathcal{I}$ we have the relationship*

$$D(\mathcal{I}) \ = \ W(\mathcal{C}) + B(\mathcal{C}) \ \geq \ \max\left\{W(\mathcal{C}), B(\mathcal{C})\right\}.$$

This indicates that, in general, we try to find the partition $(\mathcal{I}_k)_{k \in \mathcal{K}}$ of $\mathcal{I}$ that gives a minimal aggregate within-cluster dissimilarity $W(\mathcal{C})$, because this partition (based on K parameters) maximally explains the observations. We consider the single terms of $W(\mathcal{C})$ and define the (individual) within-cluster dissimilarities by

$$D(\mathcal{I}_k) = \frac{1}{2n_k} \sum_{i,l \in \mathcal{I}_k} \sum_{j=1}^{J} w_j \left(x_{i,j} - x_{l,j}\right)^2 \ = \ \sum_{j=1}^{J} w_j \sum_{i \in \mathcal{I}_k} \left(x_{i,j} - \bar{x}_{j|k}\right)^2.$$

An easy consequence is

$$D(\mathcal{I}) \ = \ \sum_{j=1}^{J} w_j \sum_{i=1}^{I} \left(x_{i,j} - \bar{x}_j\right)^2 \ \geq \ W(\mathcal{C}) \ = \ \sum_{k=1}^{K} D(\mathcal{I}_k).$$

This explains the idea that we will pursue, namely, find the K-partition of $\mathcal{I}$ that leads to a minimal aggregate within-cluster dissimilarity $W(\mathcal{C})$ in (8.6), i.e. and a maximal between-cluster similarity $B(\mathcal{C})$. This optimally describes the observed heterogeneity on the portfolio considered.

This optimization problem is of very similar nature as the ones studied above, for instance, the regression tree construction problem of Chapter 6. We try to solve a global minimization problem that typically is not tractable do to computational constraints. Therefore, we present an algorithm in the next section which (at least) converges to a local minimum of the objective function.

8.2.3 K-means clustering algorithm

In this section we describe the K-means clustering algorithm, see Algorithm 10.1 in James et al. [74], which provides a classifier $\mathcal{C}$ on $\mathcal{I}$ for a fixed number K of categorical classes. This K-means clustering algorithm is very popular (and it is probably the most used one for solving this kind of classification problem).
For a given classifier $\mathcal{C} : \mathcal{I} \to \mathcal{K}$ we consider the aggregate within-cluster dissimilarity given by, see (8.6),

$$W(\mathcal{C}) \;=\; \sum_{k=1}^{K} \sum_{j=1}^{J} w_j \sum_{i \in \mathcal{I}_k} \Big(x_{i,j} - \bar{x}_{j|k} \Big)^2,$$

with empirical means on $\mathcal{I}_k$ given by $\bar{x}_{j|k} = n_k^{-1} \sum_{i \in \mathcal{I}_k} x_{i,j}$.

The optimal classifier $\mathcal{C}^* : \mathcal{I} \to \mathcal{K}$ for given K is found by solving

$$\mathcal{C}^* \;=\; \operatorname*{argmin}_{\mathcal{C}:\mathcal{I}\to\mathcal{K}} W(\mathcal{C}) \;=\; \operatorname*{argmin}_{\mathcal{C}:\mathcal{I}\to\mathcal{K}} \min_{(m_{j|k})_{j,k}} \sum_{k=1}^{K} \sum_{j=1}^{J} w_j \sum_{i:\mathcal{C}(i)=k} \Big(x_{i,j} - m_{j|k} \Big)^2.$$

This optimal classifier is approximated by alternating the two minimization steps.

K-Means Clustering Algorithm.

---

(0) Choose an initial classifier $\mathcal{C}^{(0)} : \mathcal{I} \to \mathcal{K}$ with corresponding empirical means $(\bar{x}^{(0)}_{j|k})_{j,k}$.

(1) Repeat for $t \geq 1$ until no changes are observed:

(a) given the present empirical means $(\bar{x}^{(t-1)}_{j|k})_{j,k}$ choose the classifier $\mathcal{C}^{(t)} : \mathcal{I} \to \mathcal{K}$ such that for each driver $i \in \mathcal{I}$ we have

$$\mathcal{C}^{(t)}(i) \;=\; \operatorname*{argmin}_{k \in \mathcal{K}} \sum_{j=1}^{J} w_j \Big(x_{i,j} - \bar{x}^{(t-1)}_{j|k} \Big)^2;$$

(b) calculate the empirical means $(\bar{x}^{(t)}_{j|k})_{j,k}$ on the new partition induced by classifier $\mathcal{C}^{(t)} : \mathcal{I} \to \mathcal{K}$.

Remarks 8.4.

- The K-means clustering algorithm converges: Note that due to the minimization in step (1a) and due to Lemma 8.2 for step (1b) each iteration in (1) reduces the aggregate within-cluster dissimilarity, i.e., we have

$$W(\mathcal{C}^{(0)}) \;\geq\; \ldots \;\geq\; W(\mathcal{C}^{(t-1)}) \;\geq\; W(\mathcal{C}^{(t)}) \;\geq\; \ldots \;\geq\; 0.$$

  This provides convergence in finite time because we have finitely many K-partitions. However, we may end up in a local minimum. Therefore, one may use different (random) initial classifiers (seeds) $\mathcal{C}^{(0)}$ in step (0) of the algorithm to back-test the solution.

- Another issue is the choice of the constant K for the number of clusters considered. We may start with running the algorithm for $K = 2$ which leads to a binary partition $(\mathcal{I}_k)_{k=1,2}$ with empirical means $(\bar{x}_{j|k})_{j,k=1,2}$. For $K = 3$, we may then use these empirical means $(\bar{x}_{j|k})_{j,k=1,2}$ together with $(\bar{x}_j)_j$ as initial values for the K-means clustering algorithm with $K = 3$. This choice ensures that the resulting aggregate within-cluster dissimilarity is decreasing in K. This is exactly what we consider in Listing 8.1, below.

- For an arbitrary driver $\boldsymbol{x} \in \mathcal{P}_J \supset \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ we extend classifier $\mathcal{C}^{(t)}$ of the above algorithm to

$$\boldsymbol{x} \;\mapsto\; \mathcal{C}^{(t)}(\boldsymbol{x}) \;=\; \underset{k\in\mathcal{K}}{\operatorname{argmin}} \sum_{j=1}^{J} w_j \left(x_j - \bar{x}_{j|k}^{(t-1)}\right)^2,$$

  where $(\bar{x}_{j|k}^{(t-1)})_{j,k}$ are the empirical means obtained after the $(t-1)$-st iteration of the K-means clustering algorithm.

8.2.4 Example

We apply the K-means clustering algorithm to an example. We therefore choose the synthetic data which is generated by the simulation machine available from:

`https://people.math.ethz.ch/~wmario/Simulation_Machines/simulation.machine.heatmaps.V1.zip`

We exactly use the set-up proposed in that simulation machine for $n = 2'000$ car drivers. This generates heatmaps of grid size 20×20, i.e. with $J = 400$. Moreover, we set $w \equiv 1$.

Listing 8.1: K-means algorithm for $K = 2, \ldots, 10$

```
1  J           <- 400
2  WCSS        <- array(NA, c(10))
3  WCSS[1]     <- nrow(X) * sum(colSds(as.matrix(X))^2)
4  Classifier  <- array(1, c(10, nrow(X)))
5
6  set.seed(100)
7  for (K in 2:10){
8     if (K==2){(K_res <- kmeans(X,K) )}
9     if (K>2){(K_res  <- kmeans(X,K_centers) )}
10    clusters         <- K_res$cluster
11    WCSS[K]          <- sum(K_res$withins)
12    Classifier[K,]   <- clusters
13    K_centers        <- array(NA, c(K+1, J))
14    K_centers[K+1,]  <- colMeans(X)
15    K_centers[1:K,]  <- K_res$centers
16                 }
```

In Listing 8.1 we provide the R code of the K-means algorithm for $K = 2, \ldots, 10$, where we always use the resulting centers for given K as initial values for the algorithm with $K + 1$ centers (line 14) and the new center is initialized by the overall mean $(\bar{x}_j)_j$ (line 13). In Figure 8.5 we provide the results of iterations $K = 1, \ldots, 4$ (rows 1-4).
In Figure 8.6 we provide the resulting decrease in aggregate within-cluster dissimilarities $W(\mathcal{C})$ as a function of $K = 1, \ldots, 10$. We observe that the first split is by far the most efficient one, and this is supported by rows 1 and 2 in Figure 8.5, where we see that

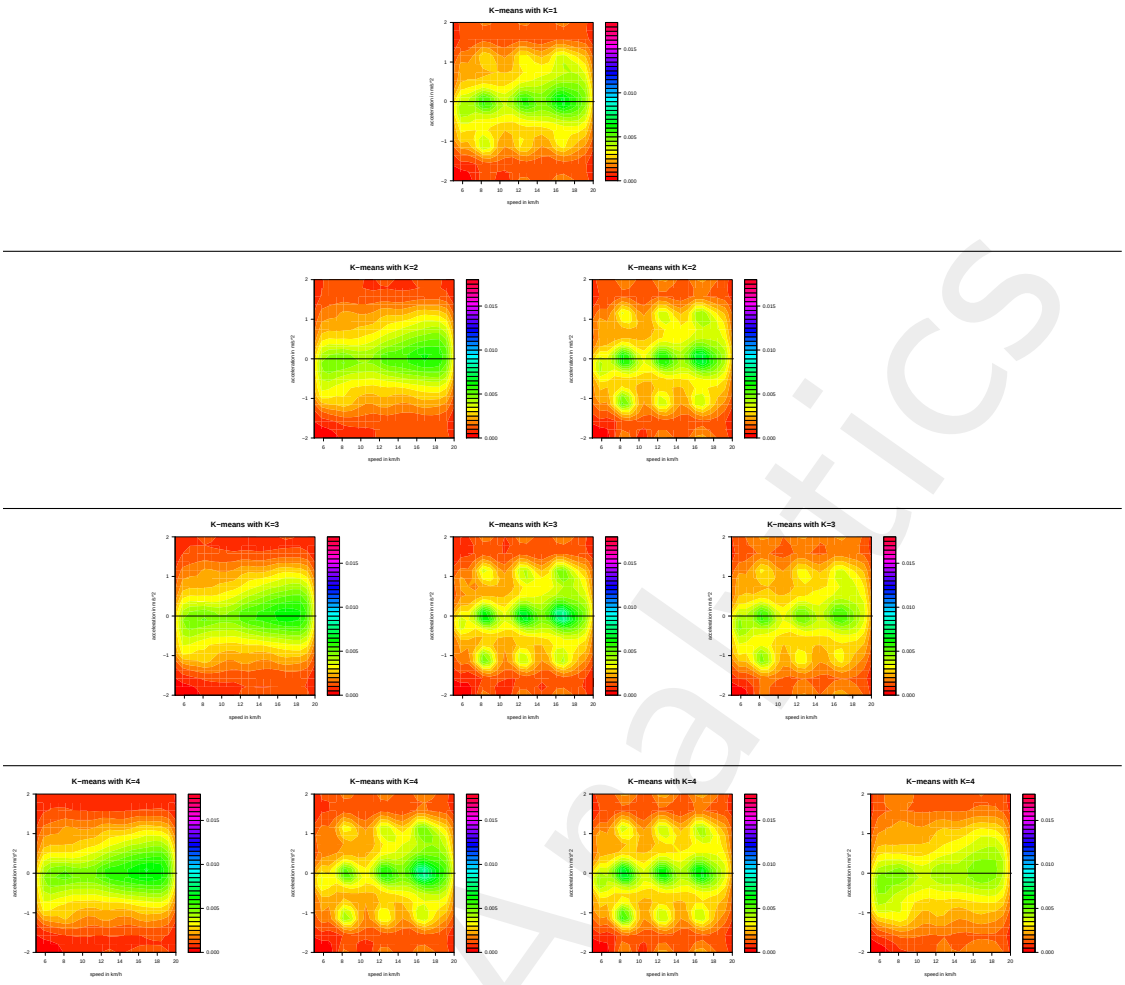


Figure 8.5: K-means clustering algorithm: resulting empirical means $(\bar{x}_{j|k})_{j,k=1,\ldots,K}$ for the different constants $K = 1, 2, 3, 4$ corresponding to rows 1-4.

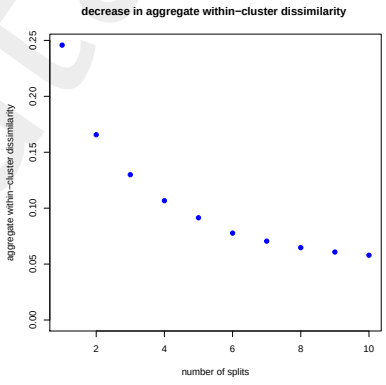


Figure 8.6: Aggregate within-cluster dissimilarities $W(\mathcal{C})$ as a function of $K = 1, \ldots, 10$.

smooth plots are separated from plots with multiple local maxima. The second split

from $K = 2$ to $K = 3$ further splits the plots with multiple local maxima. The third split then separates more smooth v-a heatmaps (bottom-right) from the other heatmaps. Remark that individual policies may change clusters from K to $K + 1$ in Figure 8.5. Based on a fixed K for the number of labels, and depending on the chosen dissimilarity measure this provides us with categorical feature information that may be used in car frequency prediction (if we believe that the resulting clusters describe common driving styles and risk factor behaviors).

8.3 Principal component analysis

The K-means clustering approach described in the previous section has the disadvantage that it leads to (nominal) categorical classes. We have seen that this may be a disadvantage in a claims frequency regression analysis because it may induce that many regression parameters are necessary (if we use, for instance, dummy coding). In this section we introduce dimension reduction techniques that lead to continuous (low-) dimensional representations of v-a heatmaps.

We denote by $\mathfrak{X} = (\boldsymbol{x}_1', \ldots, \boldsymbol{x}_n')' \in \mathbb{R}^{n \times J}$ the $n \times J$ design matrix that contains the v-a heatmaps $\boldsymbol{x}_i \in \mathcal{P}_J$ of all drivers $i = 1, \ldots, n$ (we assume $n > J$). Denote by $X \in \mathbb{R}^{n \times J}$ a normalized version of $\mathfrak{X}$ so that all column means are zero and have unit variance. The goal is to find a matrix $X_q \in \mathbb{R}^{n \times J}$ of rank $q < \min\{J, n\}$ such that the least-squares reconstruction error to X is minimized. This then implies that we receive a q-dimensional representation of X. As described in Section 14.5 of Hastie et al. [62], this means that we need to perform a principal component analysis (PCA) that provides a best linear approximation to X of rank q.[2] The corresponding solutions are obtained by the following singular value decomposition (SVD) which says that there exists an $n \times J$ orthogonal matrix U ($U'U = \mathbb{1}_J$), a $J \times J$ orthogonal matrix V ($V'V = \mathbb{1}_J$) and a $J \times J$ diagonal matrix $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_J)$ with singular values $\lambda_1 \geq \ldots \geq \lambda_J \geq 0$ such that

$$X = U \Lambda V',$$

see (14.51) in Hastie et al. [62]. Multiplying from the right with V, we receive the principal components of X as the columns of the matrix

$$XV = U\Lambda = U\text{diag}(\lambda_1, \ldots, \lambda_J). \tag{8.7}$$

In PCA we now analyze how many singular values $\lambda_1 \geq \ldots \geq \lambda_q$, $1 \leq q \leq J$, and principal components (columns of $XV = U\Lambda$) we need to find good approximations to the true matrix X. The first q principal components are received from the first q columns of V called right-singular vectors of X, see (8.7).
We use the R command `svd` to receive the SVD. In Figure 8.7 we provide the resulting singular values $\lambda_1 \geq \ldots \geq \lambda_J \geq 0$ on the original scale and on the log scale. We

[2] In a nutshell, a PCA provides an orthogonal transformation to a system of coordinates such that the projections onto these coordinates provide the directions of the biggest variances of the design matrix X (in decreasing order and always orthogonal to the previous ones). If we only keep the first q principal components, then we receive a rank q approximation X_q to X that has minimal least-squares reconstruction error, i.e. we project from dimension J to dimension q at a minimal loss of information.

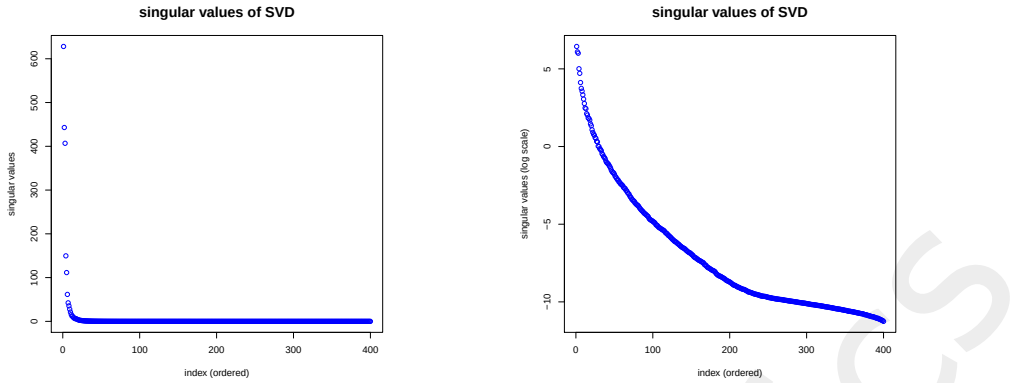


Figure 8.7: Singular values $\lambda_1 \geq \ldots \geq \lambda_J \geq 0$ with $J = 400$ on the original scale (lhs) and on the log scale (rhs).

observe that the first three singular values are dominant, thus, reduce the reconstruction error most. Denote $\Lambda_q = \text{diag}(\lambda_1, \ldots, \lambda_q, 0, \ldots, 0) \in \mathbb{R}^{J \times J}$. The best possible rank q approximation to X (i.e. with minimal least-squares reconstruction error) is given by

$$X_q = U \Lambda_q V',$$

and doing the back-transformation to the original column means and variances, respectively, we obtain the rank q approximation $\mathfrak{X}_q$ to $\mathfrak{X}$.

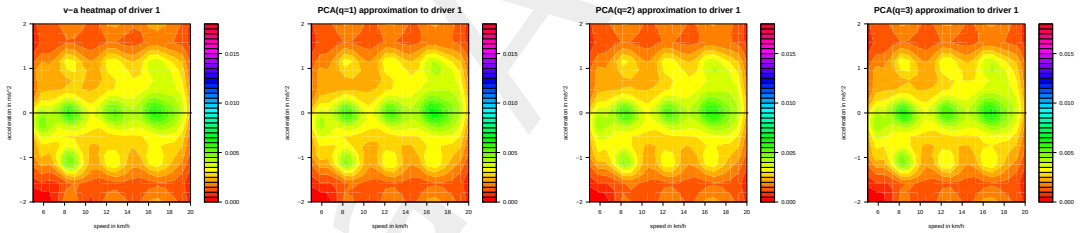


Figure 8.8: PCA approximation of the v-a heatmap of driver 1: (lhs) original heatmap $\boldsymbol{x}_1$, (other columns) approximations for $q = 1, \ldots, 3$.

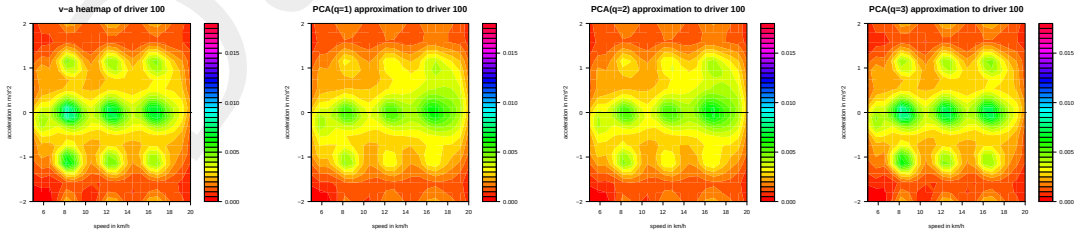


Figure 8.9: PCA approximation of the v-a heatmap of driver 100: (lhs) original heatmap $\boldsymbol{x}_{100}$, (other columns) approximations for $q = 1, \ldots, 3$.

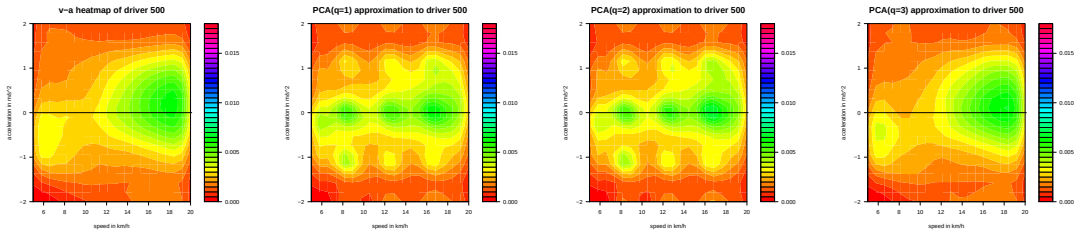


Figure 8.10: PCA approximation of the v-a heatmap of driver 500: (lhs) original heatmap $\boldsymbol{x}_{500}$, (other columns) approximations for $q = 1, \ldots, 3$.

In Figures 8.8-8.10 we illustrated the three different drivers $i = 1, 100, 500$. The first plot in each figure shows the original v-a heatmap $\boldsymbol{x}_i$ of that driver, and columns 2-4 show the PCA approximations for $q = 1, 2, 3$ (rows $i = 1, 100, 500$ of matrix $\mathfrak{X}_q$). Just by looking at these plots we claim that we need (at least) three principal components to represent our v-a heatmaps sufficiently accurately.

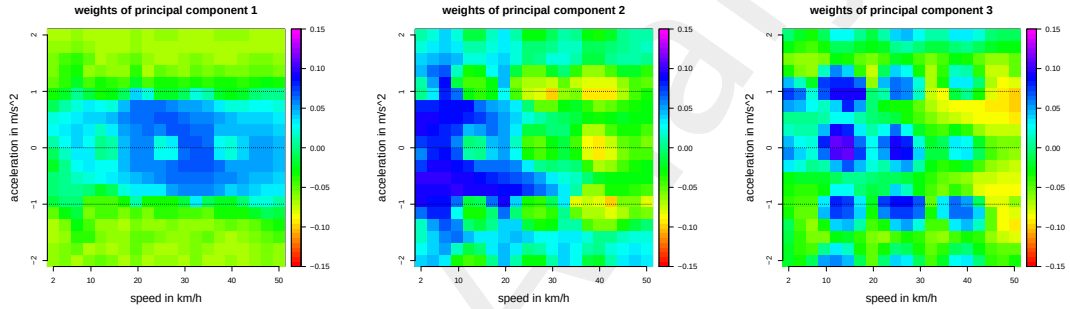


Figure 8.11: First three right-singular vectors of X, i.e. first three columns of V.

In Figure 8.11 we illustrate the first three right-singular vectors of X, i.e. the first three columns of V. These provide the first three principal components, see (8.7). We interpret these figures as follows: the first right-singular vector of X measures strong acceleration and deceleration (beyond ± 1 m/s^2), the second right-singular vector balances differences between high and low velocities in R, and the third right-singular vector takes care of local maxima.
In Figure 8.12 we provide the first three principal components, i.e. we consider the projection of $\mathfrak{X}$ onto the first three principal components (setting $q = 3$). The dots in Figure 8.12 are colored blue-red-green according to the K-means classification received in Figure 8.5 for $K = 3$. We observe that the PCA with $q = 3$ and the K-means clustering with $K = 3$ basically come to the same results because the colored dots from the K-means clustering seem well separated in the PCA representation, see Figure 8.12.

Remarks.

- The PCA leads to a rank q approximation $\mathfrak{X}_q$ to $\mathfrak{X}$. This approximation is a linear approximation that minimizes the least-square reconstruction error $\|X_q - X\|_2$. The

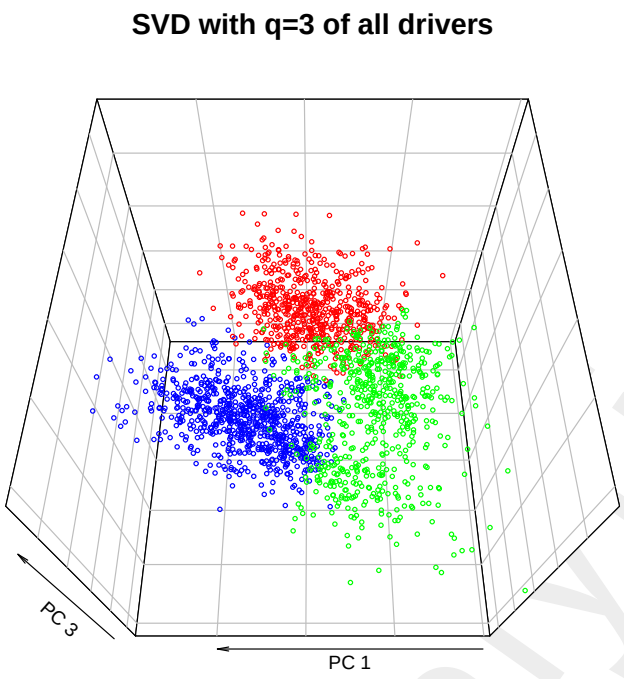


Figure 8.12: First three principal components representation of all drivers, the colors (blue-red-green) illustrate the K-means classes for $K = 3$ received in Figure 8.5.

PCA has the disadvantage that the resulting approximations are not necessarily v-a heatmaps, for instance, the linearity implies that some entries of $\mathfrak{X}_q$ may be negative, violating the assumption of having probabilities $\boldsymbol{x}_q \in \mathcal{P}_J$.

- More generally, we may consider so-called auto-encoders. An auto-encoder is a composition $\pi = \psi \circ \varphi$ of two maps. The encoder is given by

$$\varphi : \mathcal{X} \to \mathcal{Z}, \qquad \boldsymbol{z} = \varphi(\boldsymbol{x}),$$

and decoder is given by

$$\psi : \mathcal{Z} \to \mathcal{X}, \qquad \boldsymbol{x} = \psi(\boldsymbol{z}).$$

A good auto-encoder considers the two maps φ and ψ such that $\pi(\boldsymbol{x}) = (\psi \circ \varphi)(\boldsymbol{x}) \approx \boldsymbol{x}$, that is, the output signal $\pi(\boldsymbol{x})$ cannot be distinguished from the input signal $\boldsymbol{x}$. In that case we can use the value $\boldsymbol{z} = \varphi(\boldsymbol{x}) \in \mathcal{Z}$ as representation for $\boldsymbol{x}$, because the decoder ψ allows us to reconstruct the input signal $\boldsymbol{x}$ from its encoded value $\boldsymbol{z}$. If the dimension of $\mathcal{Z}$ is low we receive a low-dimensional representation of $\boldsymbol{x} \in \mathcal{X}$.

- Bottleneck neural networks are popular architectures of auto-encoders, we refer to Kramer [83] and Hinton–Salakhutdinov [65]. For a bottleneck neural network we

consider a feed-forward neural network encoder

$$\varphi : \mathbb{R}^{q_0} \to \mathbb{R}^{q_m}, \qquad \varphi(\boldsymbol{x}) = \left(\boldsymbol{z}^{(m)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}),$$

and feed-forward neural network decoder

$$\psi : \mathbb{R}^{q_m} \to \mathbb{R}^{q_{2m}}, \qquad \psi(\boldsymbol{z}) = \left(\boldsymbol{z}^{(2m)} \circ \cdots \circ \boldsymbol{z}^{(m+1)}\right)(\boldsymbol{z}),$$

with dimensions $q_0 = q_{2m} \gg q_m$. This network is then trained such that we have $\pi(\boldsymbol{x}) = (\psi \circ \varphi)(\boldsymbol{x}) \approx \boldsymbol{x}$, with q_m-dimensional representation $\boldsymbol{z} = \varphi(\boldsymbol{x})$ of $\boldsymbol{x}$. Since we typically assume $q_m \ll q_0 = q_{2m}$, this neural network architecture is called bottleneck neural network with bottleneck activations $\boldsymbol{z} = \varphi(\boldsymbol{x}) \in \mathbb{R}^{q_m}$. Sometimes, this low-dimensional representation is also called non-linear PCA.

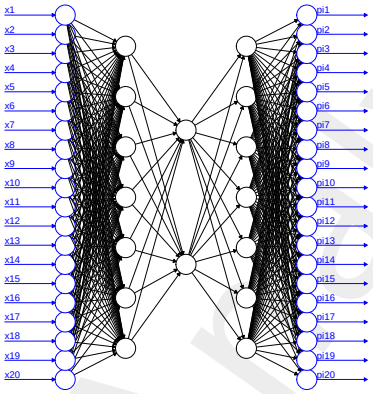


Figure 8.13: Bottleneck neural network with bottleneck $q_2 = 2$ and $q_0 = q_4 = 20$.

In Figure 8.13 we illustrate a bottleneck neural network. Input and output dimensions are $q_0 = q_4 = 20$ (blue color), and the bottleneck dimension is $q_2 = 2$. Observe that the chosen neural network architecture in Figure 8.13 is symmetric w.r.t. the bottleneck. This has been done on purpose because this symmetry is essential for finding good calibrations, we refer to Hinton–Salakhutdinov [65].

- There remains to prove in a comprehensive study that these low dimensional representations of the v-a heatmaps have explanatory power for claims frequency predictions. First attempts on rather small portfolios have been done in [49, 52].
- Another direction that we may pursue is to directly analyze individual trips, and score these individual trips, see also Figure 8.2. This individual trip scoring is related to time series analysis, and the methods at hand are convolutional neural networks (for fixed length of the time series), long short-term memory (LSTM) networks or gated recurrent unit (GRU) networks. Preliminary analysis in [51] indicates that 180 sec of driving experience already allows us to allocate randomly chosen trips to the right drivers.

Appendix A

Motor Insurance Data

A.1 Synthetic data generation

We consider a motor third party liability (MTPL) insurance portfolio that consists of $n = 500'000$ car insurance policies. This portfolio has been constructed synthetically based on the French MTPL data `freMTPL2freq`.[1] In particular, we have been transforming the old French regions to Swiss cantons by meeting related population densities.

Listing A.1: Synthetic MTPL insurance portfolio

```
1  'data.frame':   500000 obs. of  10 variables:
2   $ id     : int  1 2 3 4 5 6 7 8 9 10 ...
3   $ expo   : num  0.33 0.08 0.92 1 0.63 1 0.13 1 1 0.85 ...
4   $ age    : int  66 31 60 66 63 53 61 41 41 39 ...
5   $ ac     : int  4 1 6 4 3 5 13 11 4 6 ...
6   $ power  : int  3 7 5 2 5 3 4 1 4 1 ...
7   $ gas    : Factor w/ 2 levels "Diesel","Regular": 2 1 1 1 2 1 2 1 1 1 ...
8   $ brand  : Factor w/ 11 levels "B1","B10","B11",..: 4 1 1 1 4 1 7 7 1 7 ...
9   $ area   : Factor w/ 6 levels "A","B","C","D",..: 2 1 3 3 2 3 3 2 4 4 ...
10  $ dens   : int  83 34 223 283 74 125 323 65 533 889 ...
11  $ ct     : Factor w/ 26 levels "ZH","AG","AI",..: 5 6 2 8 25 26 2 24 5 10 ...
```

In Listing A.1 we provide a short summary of the synthetic portfolio. For each of these car insurance policies we have feature information (on lines 4-11 of Listing A.1) and the corresponding years at risk (`expo`) information $v_i \in (0, 1]$ (on line 3 of Listing A.1). Line 2 of Listing A.1 gives the policy number (`id`). Since the policy number is not considered to be of explanatory character for claims prediction we drop this information from all our considerations.

We start by describing the exposure. The years at risk information `expo` is illustrated in Figure A.1 (lhs and middle). For the years at risk we have the following properties $\min_i v_i = 0.02$ and $\max_i v_i = 1$, that is, the minimal observed time insured in our portfolio is 7.3 days and the maximal time insured is 1 year. The average time insured is $\sum_i v_i/n = 0.506$, which corresponds to 185 days, the median time insured is 172 days, and (only) 21% of the policies are insured during the whole year.

[1] The data `freMTPL2freq` is included in the `R` package `CASdatasets`, see Charpentier [25]. It is described on page 55 of the reference manual [23], we also refer to the CASdatasets website `http://cas.uqam.ca`.

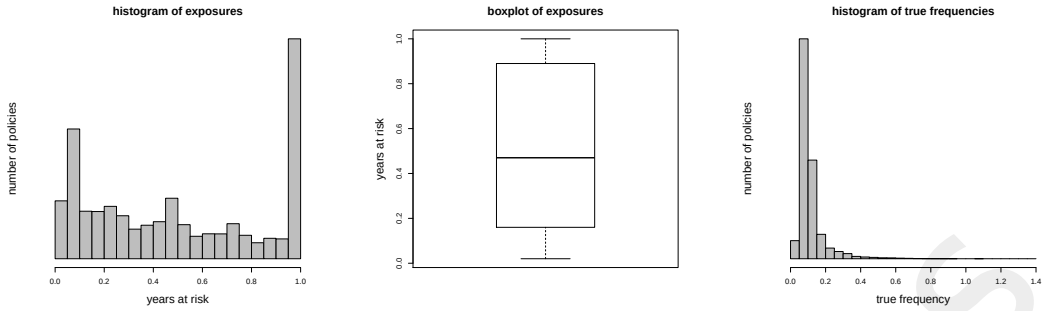


Figure A.1: (lhs) Histogram and (middle) boxplot of the years at risk `expo` of the $n = 500'000$ car insurance policies, (rhs) histogram of the true frequencies $\lambda^\star(\boldsymbol{x}_i)$ of all insurance policies $i = 1, \ldots, n$.

Next, we describe the feature information provided on lines 4-11 of Listing A.1:

`age`	age of driver, continuous feature in $\{18, \ldots, 90\}$ years;
`ac`	age of car, continuous feature in $\{0, \ldots, 35\}$ years;
`power`	power of car, continuous feature in $\{1, \ldots, 12\}$;
`gas`	fuel type of car (diesel/regular petrol), binary feature;
`brand`	brand of car, categorical feature with 11 labels;
`area`	area code, categorical feature with 6 labels;
`dens`	density at the living place of the driver, continuous feature in $[1, 27'000]$;
`ct`	Swiss canton of the car license plate, categorical feature with 26 labels.

(A.1)

In Figure A.2 we illustrate the 26 Swiss cantons.

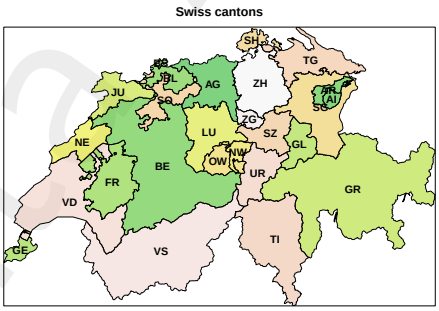


Figure A.2: Swiss cantons.

Based on this feature information we design a synthetic regression function $\boldsymbol{x}_i \mapsto \lambda^\star(\boldsymbol{x}_i)$ from which we sample our claims observations N_i for $1 \leq i \leq n$. In the first step we need to define a feature space $\mathcal{X}^\star$ containing the 8 explanatory variables given in (A.1). Note that this requires transformation of the categorical feature components into an appropriate form, for instance, by dummy coding. In the second step we design a "realistic" (true) expected frequency function and denote it by

$$\lambda^\star : \mathcal{X}^\star \to \mathbb{R}_+, \qquad \boldsymbol{x} \mapsto \lambda^\star(\boldsymbol{x}).$$

The choice we make is "realistic" in the sense that it reflects typical characteristics of a real car insurance portfolio. Deliberately we do not provide the specific functional form of that regression function $\lambda^\star$ here, but we leave it to the reader to search for the true expected frequency function. On line 12 of Listing A.2 (denoted by `truefreq`) we provide the true expected frequencies $\lambda^\star(\boldsymbol{x}_i)$ of all insurance policies $i = 1, \ldots, n$. The histogram of these true expected frequencies is given in Figure A.1 (rhs). Note that these true expected frequencies are right-skewed.

Listing A.2: Synthetic MTPL insurance portfolio including claims

```
1  'data.frame':   500000 obs. of  12 variables:
2   $ id       : int  1 2 3 4 5 6 7 8 9 10 ...
3   $ expo     : num  0.33 0.08 0.92 1 0.63 1 0.13 1 1 0.85 ...
4   $ age      : int  66 31 60 66 63 53 61 41 41 39 ...
5   $ ac       : int  4 1 6 4 3 5 13 11 4 6 ...
6   $ power    : int  3 7 5 2 5 3 4 1 4 1 ...
7   $ gas      : Factor w/ 2 levels "Diesel","Regular": 2 1 1 1 2 1 2 1 1 1 ...
8   $ brand    : Factor w/ 11 levels "B1","B10","B11",..: 4 1 1 1 4 1 7 7 1 7 ...
9   $ area     : Factor w/ 6 levels "A","B","C","D",..: 2 1 3 3 2 3 3 2 4 4 ...
10  $ dens     : int  83 34 223 283 74 125 323 65 533 889 ...
11  $ ct       : Factor w/ 26 levels "ZH","AG","AI",..: 5 6 2 8 25 26 2 24 5 10 ...
12  $ truefreq: num  0.0599 0.1192 0.0743 0.0928 0.05 ...
13  $ claims   : int  0 0 0 0 0 0 0 0 0 0 ...
```

Remarks.

- Knowing the true expected frequencies $\lambda^\star(\boldsymbol{x}_i)$ of all insurance policies $i = 1, \ldots, n$ has the (big) advantage that we can explicitly quantify the quality of all considered estimated models. However, this is not the typical situation in practice, therefore, any number that can only be calculated because we know the true model is highlighted in green color in these notes.
- Whenever we refer to the true model we use the *-sign in the notation of the feature space $\mathcal{X}^\star$ and the regression function $\lambda^\star$.
- We leave it to the reader to unravel the true expected frequency function $\lambda^\star : \mathcal{X}^\star \to \mathbb{R}_+$. It contains non-log-linear terms and non-trivial interaction terms. The data can be downloaded from

`https://people.math.ethz.ch/~wmario/Lecture/MTPL_data.csv`

Based on regression function $\lambda^\star : \mathcal{X}^\star \to \mathbb{R}_+$ we generate the observations $N_1, \ldots, N_n$ which comprises the data

$$\mathcal{D} = \{(N_1, \boldsymbol{x}_1, v_1), \ldots, (N_n, \boldsymbol{x}_n, v_n)\}. \tag{A.2}$$

This data $\mathcal{D}$ is simulated by independently generating observations N_i from

$$N_i \overset{\text{ind.}}{\sim} \text{Poi}\left(\lambda^\star(\boldsymbol{x}_i) v_i\right) \qquad \text{for } i = 1, \ldots, n. \tag{A.3}$$

The simulated `claims` are illustrated on line 13 of Listing A.2.

We briefly analyze whether the generated `claims` are a typically realization of our simulation model. The true average portfolio frequency and the empirical average portfolio frequency are given by, respectively,

$$\bar{\lambda}^\star = \frac{\sum_{i=1}^n \lambda^\star(\boldsymbol{x}_i) v_i}{\sum_{i=1}^n v_i} = 10.1991\% \qquad \text{and} \qquad \widehat{\lambda} = \frac{\sum_{i=1}^n N_i}{\sum_{i=1}^n v_i} = 10.2691\%. \tag{A.4}$$

Thus, the simulated data has a small positive bias compared to the true average portfolio frequency $\bar{\lambda}^\star$.

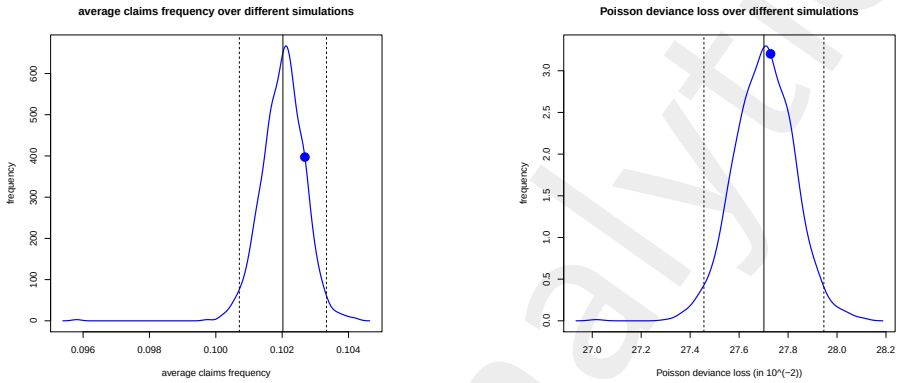


Figure A.3: (lhs) Empirical average portfolio frequencies $\widehat{\lambda}$ of 1'000 simulations with different seeds and (rhs) true Poisson deviance losses of 1'000 simulations with different seeds; the straight vertical lines show the empirical means, the dotted lines the 2 standard deviations confidence bounds, the blue dots specify the chosen simulation of Listing A.2.

We repeat 1'000 times the simulation of the data $\mathcal{D}$ in (A.2) using the true model (A.3) with different seeds. In Figure A.3 (lhs) we illustrate the empirical density of the estimated average portfolio frequency $\widehat{\lambda}$ over the 1'000 simulations with the blue dot illustrating the one chosen in Listing A.2. The sample mean of these simulations is 10.2022% with a standard deviation of 0.0657%. We note that the chosen simulation has a typical empirical average portfolio frequency $\widehat{\lambda}$, see Figure A.3 (lhs).
Next, we calculate the Poisson deviance loss w.r.t. the true model $\lambda^\star$

$$\frac{1}{n} D^*(\boldsymbol{N}, \lambda^\star) = \frac{1}{n} \sum_{i=1}^n 2 \left[\lambda^\star(\boldsymbol{x}_i) v_i - N_i - N_i \log \left(\frac{\lambda^\star(\boldsymbol{x}_i) v_i}{N_i} \right) \right] = 27.7278 \cdot 10^{-2}. \tag{A.5}$$

Also this analysis we repeat 1'000 times using different seeds for the different simulations. In Figure A.3 (rhs) we illustrate the empirical density of the Poisson deviance loss $D^*(\boldsymbol{N}, \lambda^\star)/n$ over the 1'000 simulations of observations $\boldsymbol{N}$, the blue dot again illustrates the realization chosen in Listing A.2. We observe that the chosen data has a slightly bigger value than the empirical average of $27.7013 \cdot 10^{-2}$ (straight vertical black line in Figure A.3 (rhs)), but it is within two standard deviations of the empirical average (the empirical standard deviation is $0.1225 \cdot 10^{-2}$).

A.2 Descriptive analysis

We provide a descriptive analysis of our synthetic MTPL claims data in this section.

N_i	0	1	2	3
# policies	475'153	23'773	1'012	62
in %	95.031%	4.755%	0.202%	0.012%

Table A.1: Split of the portfolio w.r.t. the number of claims.

In Table A.1 we illustrate the distribution of the observed numbers of claims $(N_i)_{1\leq i\leq n}$ across the entire portfolio of our synthetic data $\mathcal{D}$. We note that 95% of the policies do not suffer a claim. This is the so-called class imbalance problem that often causes difficulties in model calibration.

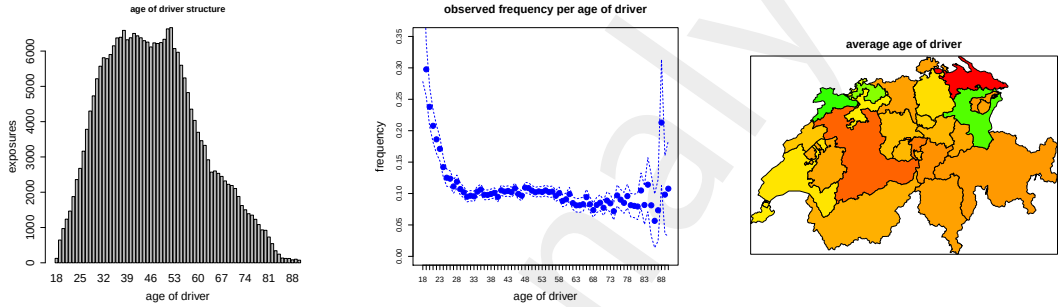


Figure A.4: Age of driver (`age`): (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) average age of driver per Swiss canton.

We provide for all 8 feature components the marginal plots in Figures A.4 to A.11. The graphs on the left-hand side show the exposure structures (in years at risk). The middle graphs provide the observed marginal frequencies (the dotted lines correspond to the estimated 2 standard deviation confidence bounds). The graphs on the right-hand side show the average feature values per Swiss canton (green color means low value and red color means high value).

From Figure A.4 (lhs) we observe that our car drivers are mostly aged between 35 and 55 years. In the middle plot of Figure A.4 we see that the observed frequency sharply drops at young ages, and it stabilizes at the age of 30. Noticeable is that we have a local maximum around the age of 50: typically, the feature `age` denotes the age of the *main* driver on a given car, but the claims frequencies may also include multiple drivers on the *same* car. The local maximum at the age of 50 is explained by the fact that at this age young adults start to drive on their parents' cars. We also note that the uncertainty is large for higher ages because we only have very few older policyholders above the age of 80 in our portfolio. Finally, Figure A.4 (rhs) shows that our portfolio is considerably old in canton TG, and rather young in cantons SG, JU and BL.

We can do a similar descriptive analysis for all other feature components. We just mention some peculiarities of our portfolio. Figure A.5: New cars have a rather high observed

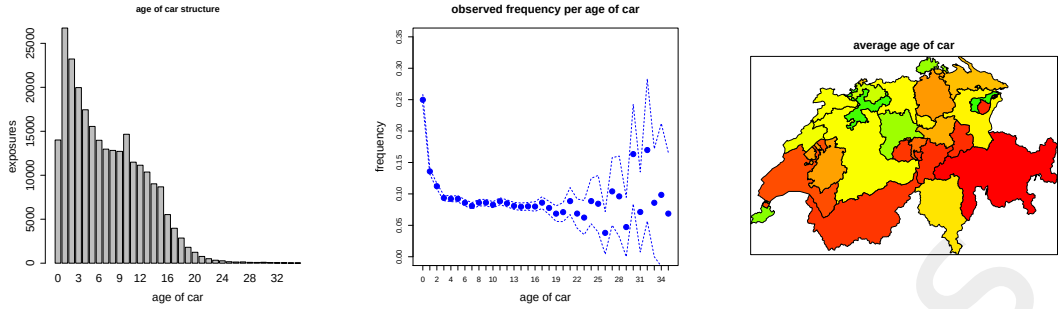


Figure A.5: Age of car (`ac`): (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) average age of car per Swiss canton.

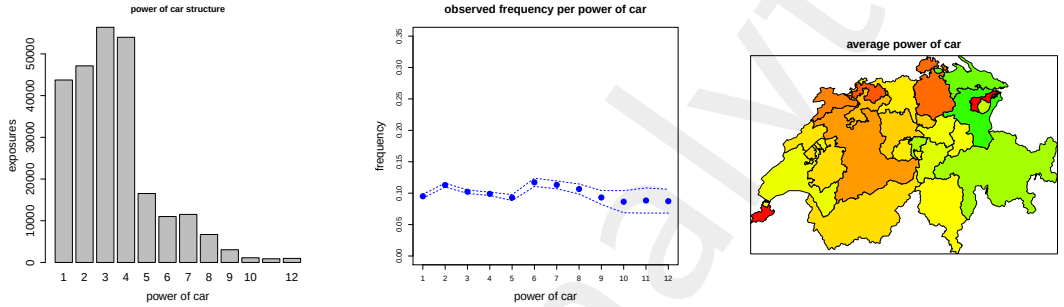


Figure A.6: Power of car (`power`) (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) average power of car per Swiss canton.

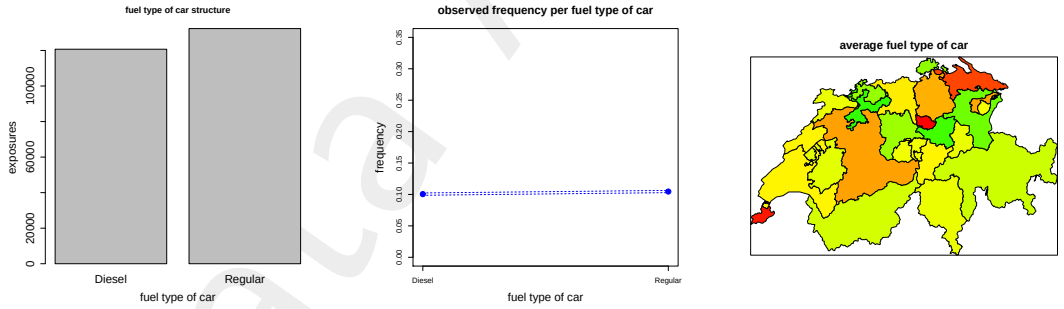


Figure A.7: Fuel type of car (`gas`) (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) ratio of regular fuel cars per Swiss canton.

claims frequency. Our portfolio has comparably old cars in many cantons that are not so densely populated (mountain area). Figures A.6 and A.7 (rhs) have some similarities because the power of a car is often related to the fuel type. From Figure A.8 we observe that car brand `B12` is rather special having a much higher observed frequency than all other car brands.

In Figures A.9 and A.10 we provide the plots of the area codes and the densities. We observe quite some similarities between these two feature components which indicates

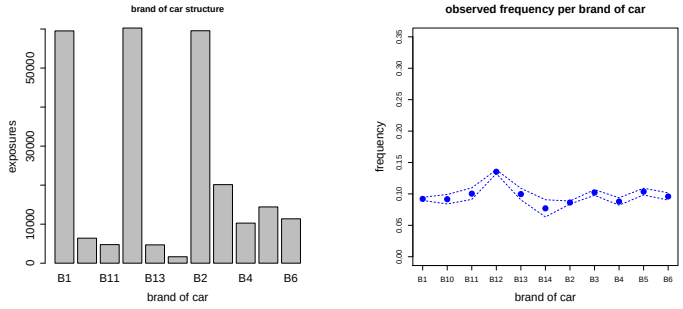


Figure A.8: Brand of car (`brand`) (lhs) portfolio structure in terms of `expo`, (rhs) observed frequency.

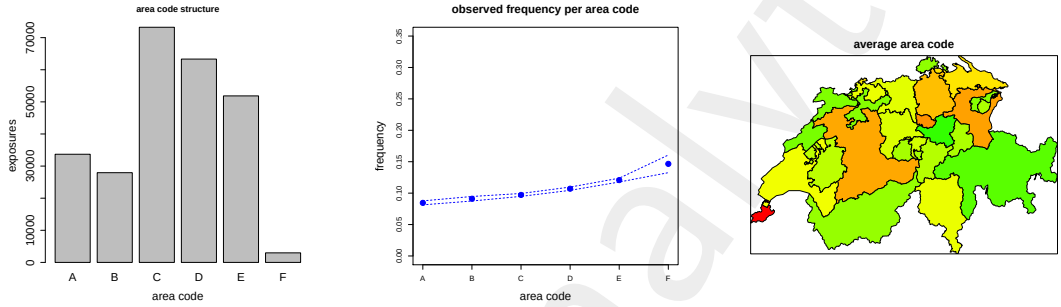


Figure A.9: Area code (`area`) (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) average area code per Swiss canton, we map $\{\text{A},\ldots,\text{F}\} \mapsto \{1,\ldots,6\}$.

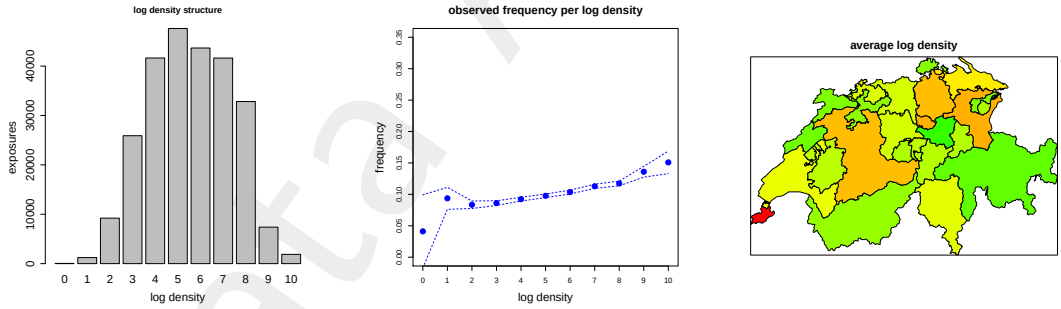


Figure A.10: Density (`dens`) (lhs) portfolio structure in terms of `expo`, (middle) observed frequency, (rhs) average density per Swiss canton.

that they are dependent. In our data the canton GE is very densely populated and observed frequencies are increasing in density. Finally, in Figure A.11 we show the empirical statistics per Swiss canton. Canton GE has the highest frequency which is likely caused by the density and the young car structure of that canton. Canton AI has the lowest frequency which needs to be explained by several different factors.[2]

[2]Note that in real Swiss MTPL data, typically, canton AI has the highest frequency which is caused by the fact that many rental car companies are located in that canton.

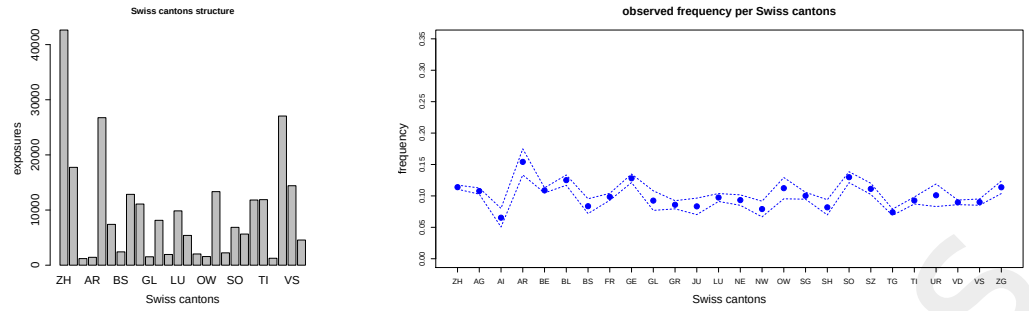


Figure A.11: Swiss cantons (`ct`) (lhs) portfolio structure in terms of `expo`, (rhs) observed frequency.

Next we analyze the dependencies between the different feature components. We therefore distinguish between categorical and continuous ones. The area code is considered to be continuous and we use the corresponding mapping $\{A,\ldots,F\} \mapsto \{1,\ldots,6\}$.

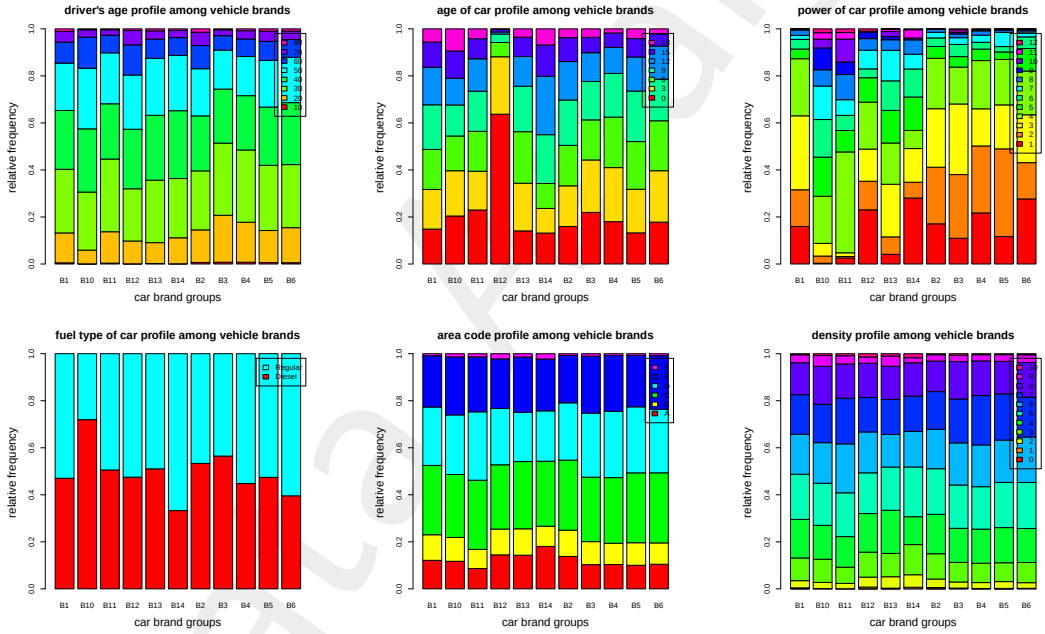


Figure A.12: Distribution of the continuous feature components across the car brands: from top-left to bottom-right: `age`, `ac`, `power`, `gas`, `area` and `dens`.

In Figure A.12 we illustrate the distribution of the continuous feature components across the car brands. Top-left: we observe that older people drive more likely car brand `B10` and younger drivers `B3`. Striking is the age of the cars of brand `B12`, more than 50% of this brand is less than 3 years old (top-middle). Cars `B10` and `B11` seem to be very powerful cars (top-right), where the former is more likely a diesel car (bottom-left).
In Figure A.13 we give the distribution of the continuous feature components across the Swiss cantons. These figures reflect (in more detail) what we have already observed in

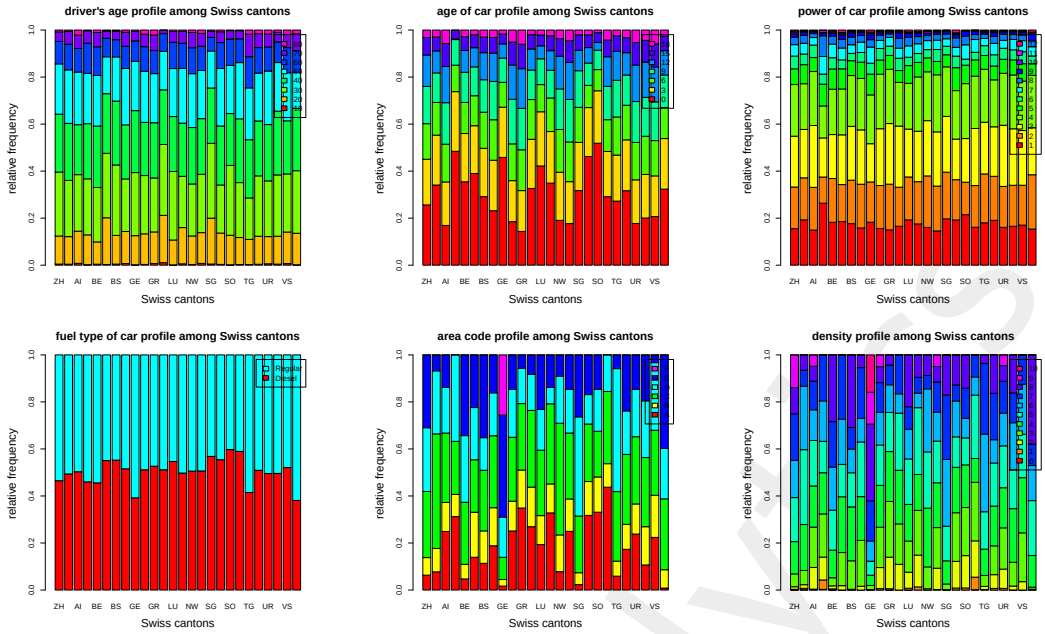


Figure A.13: Distribution of the continuous feature components across the Swiss cantons: from top-left to bottom-right: `age`, `ac`, `power`, `gas`, `area` and `dens`.

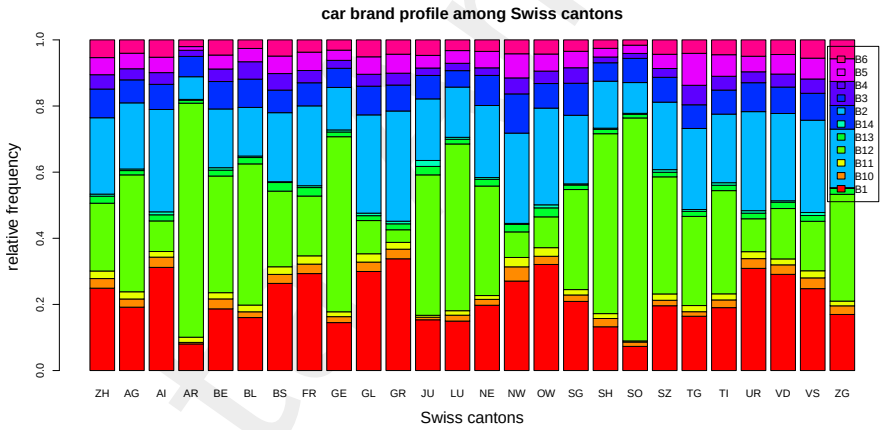


Figure A.14: Distribution of car brands across the Swiss cantons.

the Swiss maps in Figures A.4 - A.10. Canton GE plays a special role in terms of age of car, area code and density, and the mountain area AI, GL, GR, NW, OW, UR has a comparably old car portfolio (`ac`).

In Figure A.14 we study the distribution of the car brands across the Swiss cantons. We note that car brand `B12` is dominant in AR and SO. This car brand has mostly new cars which typically have a higher frequency. This may explain the high observed frequencies in these two cantons.

In Figure A.15 we provide the contour plots of the portfolio distribution of the continuous feature components. These serve us to analyze colinearity in feature components. From

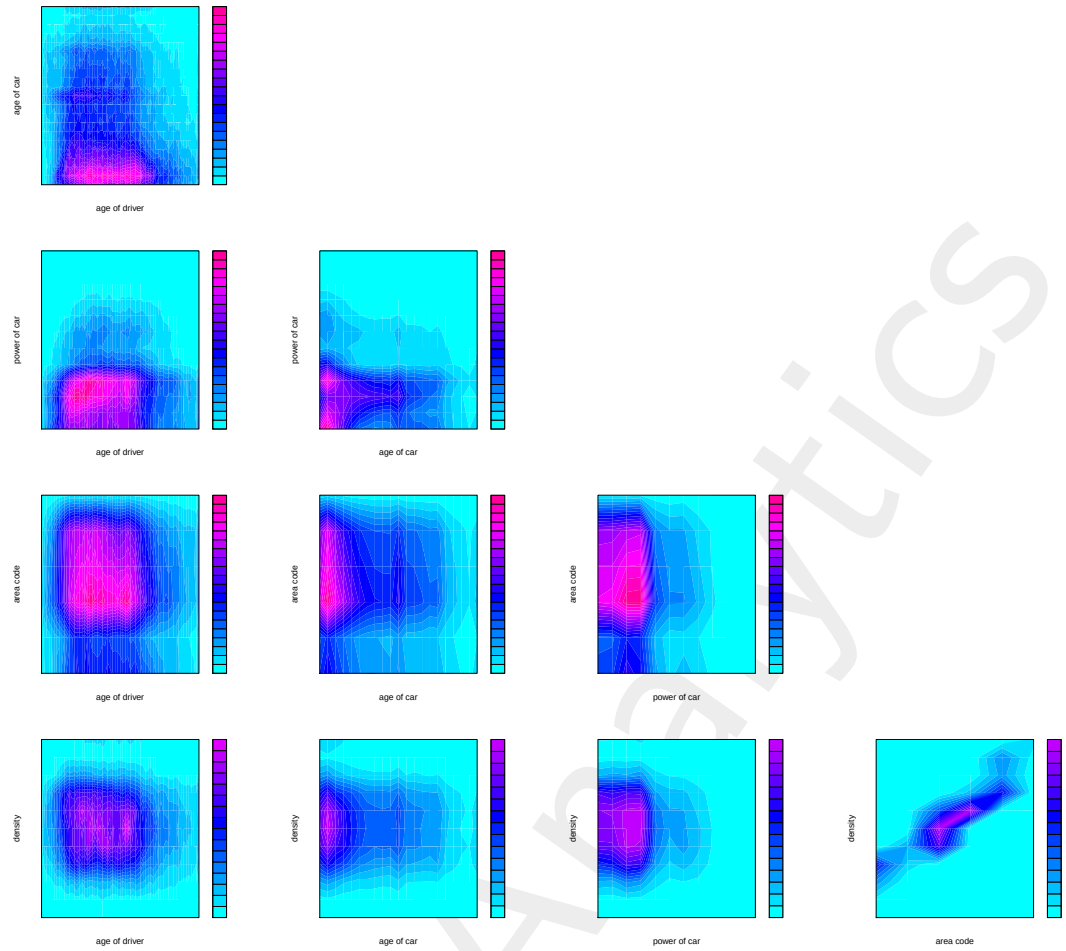


Figure A.15: Two dimensional contour plots of the portfolio distribution of the continuous features `age`, `ac`, `power`, `area` and `dens`.

these plots we cannot detect much dependence structure between the continuous feature components, except between `area` and `dens` there seems to be a strong linear relationship (we have mapped $\{A, \ldots, F\} \mapsto \{1, \ldots, 6\}$). It seems that the area code has been set according to the density of the living place of the policyholder. This has already been noticed in Figures A.9 and A.10, and it is confirmed by the corresponding correlation analysis given in Table A.2. From this we conclude that if there is such a strong linear relationship between `area` and `dens`, then likely one of the two variables is superfluous in the following regression analysis, in fact, this is what we find in some of our regression models studied. This finishes our descriptive analysis.

	age	ac	power	area	dens
age		-0.07	-0.04	-0.03	-0.01
ac	-0.09		0.00	-0.02	-0.04
power	0.05	0.01		-0.03	0.02
area	-0.03	-0.02	-0.03		**0.59**
dens	-0.03	-0.02	-0.03	**0.97**	

Table A.2: Correlations in continuous feature components: top-right shows Pearson's correlation; bottom-left shows Spearman's ρ.

Bibliography

[1] Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control* **19/6**, 716-723.

[2] Amari, S. (2016). *Information Geometry and its Applications*. Springer.

[3] Arnold, V.I. (1957). On functions of three variables. *Doklady Akademii Nauk SSSR* **114/4**, 679-681.

[4] ASTIN Big Data/Data Analytics Working Party (2015). Phase 1 Paper - April 2015. `http://www.actuaries.org`

[5] Ayuso, M., Guillen, M., Pérez-Marín, A.M. (2016). Telematics and gender discrimination: some usage-based evidence on whether men's risk of accidents differs from women's. *Risks* **4/2**, article 10.

[6] Ayuso, M., Guillen, M., Pérez-Marín, A.M. (2016). Using GPS data to analyse the distance traveled to the first accident at fault in pay-as-you-drive insurance. *Transportation Research Part C: Emerging Technologies* **68**, 160-167.

[7] Bailey, R.A. (1963). Insurance rates with minimum bias. *Proceedings CAS* **50**, 4-11.

[8] Barndorff-Nielsen, O. (2014). *Information and Exponential Families: In Statistical Theory*. John Wiley & Sons.

[9] Barndorff-Nielsen, O.E., Jensen, J.L., Kendall, W.S. (1993). *Networks and Chaos - Statistical and Probabilistic Aspects*. Chapman & Hall.

[10] Barron, A.R. (1993). Universal approximation bounds for superpositions of sigmoidal functions. *IEEE Transactions of Information Theory* **39/3**, 930-945.

[11] Billingsley, P. (1995). *Probability and Measure*. 3rd edition. Wiley.

[12] Boucher, J.-P., Côté, S., Guillen, M. (2017). Exposure as duration and distance in telematics motor insurance using generalized additive models. *Risks* **5/4**, article 54.

[13] Breiman, L. (1996). Bagging predictors. *Machine Learning* **24/2**, 123-140.

[14] Breiman, L. (2001). Random forests. *Machine Learning* **45/1**, 5-32.

[15] Breiman, L. (2001). Statistical modeling: the two cultures. *Statistical Science* **16/3**, 199-215.

[16] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). *Classification and Regression Trees*. Wadsworth Statistics/Probability Series.

[17] Breiman, L., Stone, C.J. (1978). Parsimonious binary classification trees. Technical report. Santa Monica, California: Technology Service Corporation.

[18] Bühlmann, H., Gisler, A. (2005). *A Course in Credibility Theory and its Applications*. Springer.

[19] Bühlmann, H., Straub, E. (1970). Glaubwürdigkeit für Schadensätze. *Bulletin of the Swiss Association of Actuaries* **1970**, 111-131.

[20] Bühlmann, P., Mächler, M. (2014). *Computational Statistics.* Lecture Notes. Department of Mathematics, ETH Zurich.

[21] Burges, C.J.C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* **2**, 121-167.

[22] Cantelli, F.P. (1933). Sulla determinazione empirica delle leggi di probabilità. *Giornale Dell'Istituto Italiano Degli Attuari* **4**, 421-424.

[23] CASdatasets Package Vignette (2016). Reference Manual, May 28, 2016. Version 1.0-6. Available from `http://cas.uqam.ca`.

[24] Casella, G., Berger, R.L. (2002). *Statistical Inference.* 2nd edition. Duxbury.

[25] Charpentier, A. (2015). *Computational Actuarial Science with R.* CRC Press.

[26] Chen, T., Guestrin, C. (2016). XGBoost: a scalable tree boosting system. *arXiv:*1603.02754v3.

[27] China InsurTech Lab (2017). *China InsurTech Development White Paper.* Fudan University.

[28] Congdon, P. (2006). *Bayesian Statistical Modelling.* 2nd edition. Wiley.

[29] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* **2**, 303-314.

[30] Denuit, M., Guillen, M., Trufin, J. (2018). Multivariate credibility modeling for usage-based motor insurance pricing with behavioural data. *Detralytics*, **2018-2**.

[31] Denuit, M., Maréchal, X., Pitrebois, S., Walhin, J.-F. (2007). *Actuarial Modelling of Claims Count.* Wiley.

[32] Deprez, P., Shevchenko, P.V., Wüthrich, M.V. (2017). Machine learning techniques for mortality modeling. *European Actuarial Journal* **7/2**, 337-352.

[33] Döhler, S., Rüschendorf, L. (2001). An approximation result for nets in functional estimation. *Statistics and Probability Letters* **52**, 373-380.

[34] Duane, S., Kennedy, A.D., Pendleton, B.J., Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B* **195/2**, 216-222.

[35] Efron, B. (1979). Bootstrap methods: another look at the jackknife. *Annals of Statistics* **7/1**, 1-26.

[36] Efron, B. (2020). Prediction, estimation, and attribution. *Journal of the American Statistical Association* **115/530**, 636-655.

[37] Efron, B., Hastie, T. (2016). *Computer Age Statistical Inference.* Cambridge University Press.

[38] Efron, B., Tibshirani, R.J. (1993). *An Introduction to the Bootstrap.* Chapman & Hall.

[39] Embrechts, P., Klüppelberg, C., Mikosch, T. (2003). *Modelling Extremal Events for Insurance and Finance.* 4th printing. Springer.

[40] Esteves-Booth, A., Muneer, T., Kirby, H., Kubie, J., Hunter, J. (2001). The measurement of vehicular driving cycle within the city of Edinburgh. *Transportation Research Part D: Transport and Environment* **6/3**, 209-220.

[41] Fahrmeir, L., Tutz, G. (1994). *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer.

[42] Ferrario, A., Hämmerli, R. (2019). On boosting: theory and applications. *SSRN Manuscript* ID 3402687.

[43] Ferrario, A., Noll, A., Wüthrich, M.V. (2018). Insights from inside neural networks. *SSRN Manuscript* ID 3226852.

[44] Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation* **121/2**, 256-285.

[45] Freund, Y., Schapire, R.E. (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences* **55/1**, 119-139.

[46] Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics* **29/5**, 1189-1232.

[47] Friedman, J.H. (2002). Stochastic gradient boosting. *Computational Statistics and Data Analysis* **38/4**, 367-378.

[48] Gabrielli, A., Richman, R., Wüthrich, M.V. (2020). Neural network embedding of the over-dispersed Poisson reserving model. *Scandinavian Actuarial Journal* **2020/1**, 1-29.

[49] Gao, G., Meng, S., Wüthrich, M.V. (2019). Claims frequency modeling using telematics car driving data. *Scandinavian Actuarial Journal* **2019/2**, 143-162.

[50] Gao, G., Wüthrich, M.V. (2018). Feature extraction from telematics car driving heatmap. *European Actuarial Journal* **8/2**, 383-406.

[51] Gao, G., Wüthrich, M.V. (2019). Convolutional neural network classification of telematics car driving data. *Risks* **7/1**, 6.

[52] Gao, G., Wüthrich, M.V., Yang, H. (2019). Evaluation of driving risk at different speeds. *Insurance: Mathematics & Economics* **88**, 108-119.

[53] Gilks, W.R., Richardson, S., Spiegelhalter, D.J. (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall.

[54] Glivenko, V. (1933). Sulla determinazione empirica delle leggi di probabilità. *Giornale Dell'Istituto Italiano Degli Attuari* **4**, 92-99.

[55] Gneiting, T. (2011). Making and evaluation point forecasts. *Journal of the American Statistical Association* **106/494**, 746-762.

[56] Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press, `http://www.deeplearningbook.org`

[57] Green, P.J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82/4**, 711-732.

[58] Green, P.J. (2003). Trans-dimensional Markov chain Monte Carlo. In: *Highly Structured Stochastic Systems*, P.J. Green, N.L. Hjort, S. Richardson (eds.), Oxford Statistical Science Series, 179-206. Oxford University Press.

[59] Grohs, P., Perekrestenko, D., Elbrächter, D., Bölcskei, H. (2019). Deep neural network approximation theory. Submitted to *IEEE Transactions on Information Theory* (invited paper).

[60] Hastie, T., Tibshirani, R. (1986). Generalized additive models (with discussion). *Statistical Science* **1**, 297–318.

[61] Hastie, T., Tibshirani, R. (1990). *Generalized Linear Models.* Chapman & Hall.

[62] Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* 2nd edition. Springer Series in Statistics.

[63] Hastie, T., Tibshirani, R., Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations.* CRC Press.

[64] Hastings, W.K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57/1**, 97-109.

[65] Hinton, G.E., Salakhutdinov, R.R. (2006). Reducing the dimensionality of data with neural networks. *Science* **313**, 504-507.

[66] Ho, S.-H., Wong, Y.-D., Chang, V.W.-C. (2014). Developing Singapore driving cycle for passenger cars to estimate fuel consumption and vehicular emissions. *Atmospheric Environment* **97**, 353-362.

[67] Hoffman, M.D., Gelman, A. (2014). The no-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* **15**, 1351-1381.

[68] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks* **4**, 251-257.

[69] Hornik, K., Stinchcombe, M., White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359-366.

[70] Hung, W.T., Tong, H.Y., Lee, C.P., Ha, K., Pao, L.Y. (2007). Development of practical driving cycle construction methodology: a case study in Hong Kong. *Transportation Research Part D: Transport and Environment* **12/2**, 115-128.

[71] Künsch H.R. (1993). *Mathematische Statistik.* Lecture Notes. Department of Mathematics, ETH Zurich.

[72] Ingenbleek, J.-F., Lemaire, J. (1988). What is a sports car? *ASTIN Bulletin* **18/2**, 175-187.

[73] Isenbeck, M., Rüschendorf, L. (1992). Completeness in location families. *Probability and Mathematical Statistics* **13**, 321-343.

[74] James, G., Witten, D., Hastie, T., Tibshirani, R. (2015). *An Introduction to Statistical Learning. With Applications in R.* Corrected 6th printing. Springer Texts in Statistics.

[75] Johansen, A.M., Evers, L., Whiteley, N. (2010). *Monte Carlo Methods.* Lecture Notes, Department of Mathematics, University of Bristol.

[76] Jung, J. (1968). On automobile insurance ratemaking. *ASTIN Bulletin* **5**, 41-48.

[77] Kamble, S.H., Mathew, T.V., Sharma, G.K. (2009). Development of real-world driving cycle: case study of Pune, India. *Transportation Research Part D: Transport and Environment* **14/2**, 132-140.

[78] Karush, W. (1939). *Minima of Functions of Several Variables with Inequalities as Side Constraints.* M.Sc. Dissertation. Department of Mathematics, University of Chicago.

[79] Kearns, M., Valiant, L.G. (1988). Learning Boolean formulae or finite automata is hard as factoring. Technical Report TR-14-88. Harvard University Aiken Computation Laboratory.

[80] Kearns, M., Valiant, L.G. (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery ACM* **41/1**, 67-95.

[81] Kingma, D., Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:*1412.6980.

[82] Kolmogorov, A. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR* **114/5**, 953-956.

[83] Kramer, M.A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37/2**, 233-243.

[84] Kuhn, H.W., Tucker, A.W. (1951). Nonlinear programming. *Proceedings of 2nd Berkeley Symposium.* Berkeley: University of California Press, 481-492.

[85] Latuszyński, K., Roberts, G.O., Rosenthal, J.S. (2013). Adaptive Gibbs samplers and related MCMC methods. *Annals of Applied Probability* **23/1**, 66-98.

[86] Lee, S.C.K., Lin, S. (2018). Delta boosting machine with application to general insurance. *North American Actuarial Journal* **22/3**, 405-425.

[87] Lehmann, E.L. (1983). *Theory of Point Estimation.* Wiley.

[88] Leshno, M., Lin, V.Y., Pinkus, A., Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* **6/6**, 861-867.

[89] Lorentzen, C., Mayer, M. (2020). Peeking into the black box: an actuarial case study for interpretable machine learning. *SSRN Manuscript* ID 3595944.

[90] Makavoz, Y. (1996). Random approximants and neural networks. *Journal of Approximation Theory* **85**, 98-109.

[91] Marra, G., Wood, S.N. (2011). Practical variable selection for generalized additive models. *Computational Statistics and Data Analysis* **55/7**, 2372-2387.

[92] McClure, P., Kriegeskorte, N. (2017). Representing inferential uncertainty in deep neural networks through sampling. ICLR conference paper.

[93] McCullagh, P., Nelder, J.A. (1983). *Generalized Linear Models.* Chapman & Hall.

[94] Meier, D., Wüthrich, M.V. (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. *SSRN Manuscript* ID 3656210.

[95] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics* **21/6**, 1087-1092.

[96] Montúfar, G., Pascanu, R., Cho, K., Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Neural Information Processing Systems Proceedings*$^{\beta}$ **27**, 2924-2932.

[97] Neal, R.M. (1996). *Bayesian Learning for Neural Networks.* Springer.

[98] Nelder, J.A., Wedderburn, R.W.M. (1972). Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)* **135/3**, 370-384.

[99] Nesterov, Y. (2007). Gradient methods for minimizing composite objective function. *Technical Report* **76**, Center for Operations Research and Econometrics (CORE), Catholic University of Louvain.

[100] Nielsen, M. (2017). *Neural Networks and Deep Learning.* Online book available on `http://neuralnetworksanddeeplearning.com`

[101] Noll, A., Salzmann, R., Wüthrich, M.V. (2018). Case study: French motor third-party liability claims. *SSRN Manuscript* ID 3164764.

[102] Ohlsson, E., Johansson, B. (2010). *Non-Life Insurance Pricing with Generalized Linear Models*. Springer.

[103] Park, J., Sandberg, I. (1991). Universal approximation using radial-basis function networks. *Neural Computation* **3**, 246-257.

[104] Park, J., Sandberg, I. (1993). Approximation and radial-basis function networks. *Neural Computation* **5**, 305-316.

[105] Petrushev, P. (1999). Approximation by ridge functions and neural networks. *SIAM Journal on Mathematical Analysis* **30/1**, 155-189.

[106] Pruscha, H. (2006). *Statistisches Methodenbuch: Verfahren, Fallstudien, Programmcodes*. Springer.

[107] Rentzmann, S., Wüthrich, M.V. (2019). Unsupervised learning: What is a sports car? *SSRN Manuscript* ID 3439358. Version October 14, 2019.

[108] Richman, R. (2020). AI in actuarial science - a review of recent advances - part 1. *Annals of Actuarial Science*, to appear.

[109] Richman, R. (2020). AI in actuarial science - a review of recent advances - part 2. *Annals of Actuarial Science*, to appear.

[110] Richman, R., Wüthrich, M.V. (2019). Lee and Carter go machine learning: recurrent neural networks. *SSRN Manuscript* ID 3441030.

[111] Richman, R., Wüthrich, M.V. (2020). Nagging predictors. *Risks* **8/3**, 83.

[112] Ridgeway, G. (2007). Generalized boosted models: a guide to the `gbm` package. Version of August 3, 2007.

[113] Robert, C.P. (2001). *The Bayesian Choice*. 2nd edition. Springer.

[114] Roberts, G.O., Gelman, A., Gilks, W.R. (1997). Weak convergence and optimal scaling of random walk Metropolis algorithms. *Annals of Applied Probability* **7**, 110-120.

[115] Roberts, G.O., Rosenthal, J.S. (1998). Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society, Series B* **60/1**, 255-268.

[116] Rüger, S.M., Ossen, A. (1997). The metric structure of weight space. *Neural Processing Letters* **5**, 63-72.

[117] Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature* **323/6088**, 533-536.

[118] Schapire, R.E. (1990). The strength of weak learnability. *Machine Learning* **5/2**, 197-227.

[119] Schelldorfer, J., Wüthrich, M.V. (2019). Nesting classical actuarial models into neural networks. *SSRN Manuscript* ID 3320525.

[120] Schelldorfer, J., Wüthrich, M.V. (2021). LocalGLMnet: a deep learning architecture for actuaries. *SSRN Manuscript* ID 3900350. Version August 6, 2021.

[121] Schwarz, G.E. (1978). Estimating the dimension of a model. *Annals of Statistics* **6/2**, 461-464.

[122] Shaham, U., Cloninger, A., Coifman, R.R. (2015). Provable approximation properties for deep neural networks. *arXiv:*1509.07385v3.

[123] Shmueli, G. (2010). To explain or to predict? *Statistical Science* **25/3**, 289-310.

[124] Srivastava, N., Hinton, G., Krizhevsky, A. Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning* **15**, 1929-1958.

[125] Therneau, T.M., Atkinson, E.J. (2015). An introduction to recursive partitioning using the RPART routines. *R Vignettes*, version of June 29, 2015. Mayo Foundation, Rochester.

[126] Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)* **58/1**, 267-288.

[127] Tikhonov, A.N. (1943). On the stability of inverse problems. *Doklady Akademii Nauk SSSR* **39/5**, 195-198.

[128] Valiant, L.G. (1984). A theory of learnable. *Communications of the Association for Computing Machinery ACM* **27/11**, 1134-1142.

[129] Verbelen, R., Antonio, K., Claeskens, G. (2018). Unraveling the predictive power of telematics data in car insurance pricing. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* **67/5**, 1275-1304.

[130] Wager, S., Wang, S., Liang, P.S. (2013). Dropout training as adaptive regularization. In: *Advances in Neural Information Processing Systems* **26**, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (eds.), Curran Associates, Inc, 351-359.

[131] Wang, Q., Huo, H., He, K., Yao, Z., Zhang, Q. (2008). Characterization of vehicle driving patterns and development of driving cycles in Chinese cities. *Transportation Research Part D: Transport and Environment* **13/5**, 289-297.

[132] Weidner, W., Transchel, F.W.G., Weidner, R. (2016). Classification of scale-sensitive telematic observables for riskindividual pricing. *European Actuarial Journal* **6/1**, 3-24.

[133] Weidner, W., Transchel, F.W.G., Weidner, R. (2017). Telematic driving profile classification in car insurance pricing. *Annals of Actuarial Science* **11/2**, 213-236.

[134] Wood, S.N. (2017). *Generalized Additive Models: an Introduction with* R. 2nd edition. CRC Press.

[135] Wüthrich, M.V. (2013). Non-Life Insurance: Mathematics & Statistics. *SSRN Manuscript* ID 2319328. Version March 20, 2019.

[136] Wüthrich, M.V. (2017). Covariate selection from telematics car driving data. *European Actuarial Journal* **7/1**, 89-108.

[137] Wüthrich, M.V. (2017). Price stability in regression tree calibrations. *Proceedings of CICIRCM* **2017**, Tsinghua University Press.

[138] Wüthrich, M.V. (2018). Machine learning in individual claims reserving. *Scandinavian Actuarial Journal* **2018/6**, 465-480.

[139] Wüthrich, M.V. (2020). Bias regularization in neural network models for general insurance pricing. *European Actuarial Journal* **10/1**, 179-202.

[140] Wüthrich, M.V., Merz, M. (2019). Editorial: Yes, we CANN! *ASTIN Bulletin* **49/1**, 1-3.

[141] Wüthrich, M.V., Merz, M. (2021). Statistical Foundations of Actuarial Learning and its Applications. *SSRN Manuscript* ID 3822407.

[142] Xiang, Q. (2018). *Bayesian Gaussian Random Fields Applied to Car Insurance Claim Size Modeling*. Semester Thesis. Department of Mathematics, ETH Zurich.

[143] Yukich, J., Stinchcombe, M., White, H. (1995). Sup-norm approximation bounds for networks through probabilistic methods. *IEEE Transactions on Information Theory* **41/4**, 1021-1027.

[144] Zaslavsky, T. (1975). Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Memoirs of the American Mathematical Society* **154**.

[145] Zöchbauer, P. (2016). *Data Science in Non-Life Pricing: Predicting Claims Frequencies using Tree-Based Models*. M.Sc. Thesis. Department of Mathematics, ETH Zurich.

[146] Zou, H., Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B* **67/2**, 301-320.