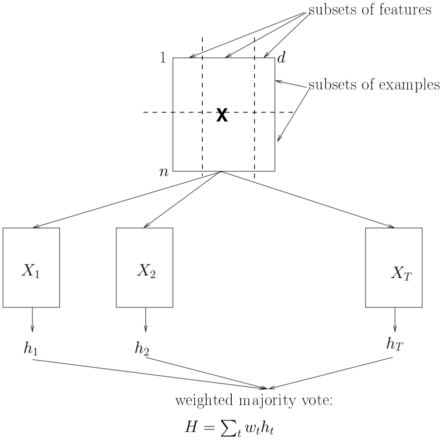
PA196: Pattern Recognition

08. Multiple classifier systems (cont'd)

Dr. Vlad Popovici
popovici@recetox.muni.cz

RECETOX
Masaryk University, Brno

General idea



Outline

Bagging (Breiman, 1996)

- bagging = bootstrap aggregation
- create T bootstrap samples X_t by *sampling with replacement*
- train a classifier on each X_t
- aggregate the classifications by plurality voting to obtain the aggregated classifier H
- a similar approach works for regression
- works well with unstable classifiers (with high variance): decision trees, neural networks

Why does bagging work?

- reduces variance (due to sampling in the test sets):

$$\begin{aligned}\mathbb{E}[(y - H(\mathbf{x}))^2] &= (y - \mathbb{E}[H(\mathbf{x})])^2 + \mathbb{E}[(H(\mathbf{x}) - \mathbb{E}[(H(\mathbf{x}))])^2] \\ &= \text{bias}^2 + \text{variance}\end{aligned}$$

- the bias of H remains approximately the same as for h_t:

$$\text{Bias}(H) = \frac{1}{T}\sum_{t=1}^{T}\text{Bias}(h_t)$$

- but the variance is reduced:

$$\text{Var}(H) \approx \frac{1}{T}\text{Var}(h_1)$$

Variants:

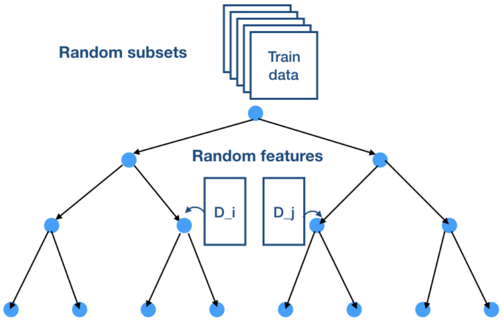
- draw random subsamples of data → "Pasting"
- draw random subsets of features → "Random Subspaces"
- draw random subsamples and random features → "Random Patches"

Outline

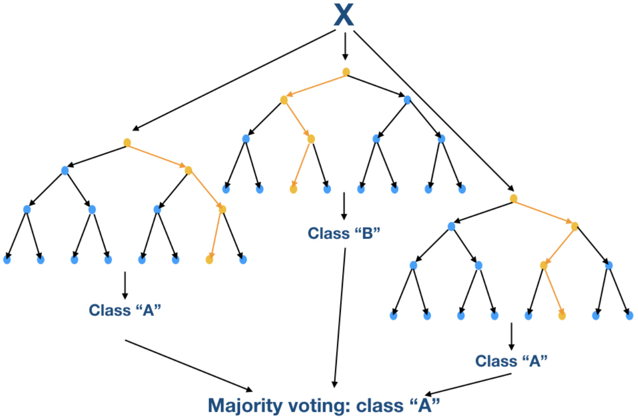
Idea: induce randomness in the base classifier (tree) and combine the predictions of an ensemble of such trees (forest) by averaging or majority vote.

- refinement of bagging trees
- (1st level of randomness) grow the trees on bootstrap samples
- (2nd level of randomness) when growing a tree, at each node consider only a random subset of features (typically $\sqrt{d}$ or $\log_2 d$ features)
- for each tree, the error rate for observation left out from the learning set is monitored ("out-of-bag" error rate)
- the result is a collection of "de-correlated" trees that by averaging/voting should lead to decreased variance of the final predictor

RF - randomness in training



RF - decision



Outline

Outline

General approach

(I will follow Freund & Schapire's tutorial on boosting)

- let $S = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, n\}$ be a data set with $y_i \in \{\pm 1\}$ and $\mathbf{x}_i$ a d–dimensional vector of features x_{ij}
- let there exist a learner (or a few) able to produce some basic classifiers h_t, based on sets such as S
- h_t will be called "weak classifiers" and the condition is that $\mathrm{Err}(h_t) = 0.5 - \epsilon_t$ where $0 < \epsilon \leq 0.5$
- for each iteration $t = 1, \ldots, T$ produce a version of the training set S_t on which h_t are fit and then, assemble their predictions

- how to select the training points at each round?
  $\rightarrow$ concentrate on most difficult points
- how to combine the weak classifiers?
  $\rightarrow$ take the (weighted) majority vote

Boosting

A general methodology of producing highly accurate predictors based on averaging some weak classifiers.

Context

- PAC framework:
  - a strong-PAC algorithm:
    - for *any* distribution (of data)
    - $\forall\epsilon > 0, \forall\delta > 0$
    - given enough data (i.i.d. from the distribution)
    - with probability at least $1 - \delta$, the algorithm will find a classifier with error $\leq \epsilon$
  - a weak-PAC algorithm: the same conditions, but the guaranteed error is $\epsilon \geq \frac{1}{2} - \gamma$
- when weak-PAC learnability leads to strong-PAC?

AdaBoost

- a development of previous "boosting" algorithms
- first to reach widespread applicability, due to simplicity of the implementation and good observed performance (in addition to theoretical performance)
- Freund & Schapire (EuroCOLT, 1995); the more complete version: "A decision-theoretic generalization of on-line learning and an application to boosting", J. Comp Sys Sc 1997
- AdaBoost: adaptive boosting

Outline

Basic AdaBoost

Input: a training set $S = \{(\mathbf{x}_i, y_i)\}$ and the number of iterations T
Output: final classifier H as a combination of weak classifiers h_t

for $t = 1$ **to** T **do**

construct a distribution D_t on $\{1, \ldots, n\}$

find a *weak classifier*

$$h_t : \mathcal{X} \to \{-1, +1\}$$

which minimizes the error ϵ_t on D_t,

$$\epsilon_t = \Pr_{D_t}[h_t(\mathbf{x}_i) \neq y_i]$$

end for

How to construct D_t?

- let $D_1(i) = 1/n$ (uninformative priors)
- given D_t and a weak classifier h_t,

$$D_{t+1} = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases} = \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

- Z_t is a properly chosen normalization constant
- $$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \in \mathbb{R}_+$$

What about the final decision/classifier?

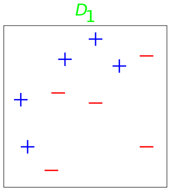
$$H(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

How to use D_t for training a classifier?

- either generate a new training sample from S by sampling according to D_t, or
- use directly the sample weights for constructing h_t

(Classical) Example (Freund & Schapire)

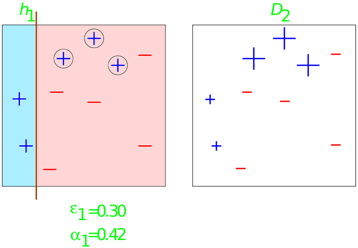
Initial state:



weak classifiers: single variable threshold function

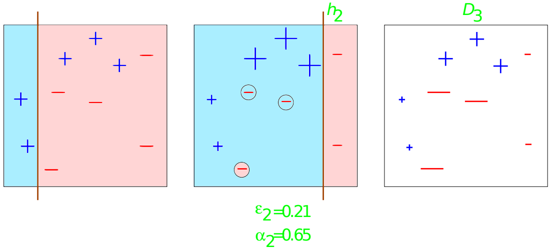
(Classical) Example (Freund & Schapire)

Iteration 1:



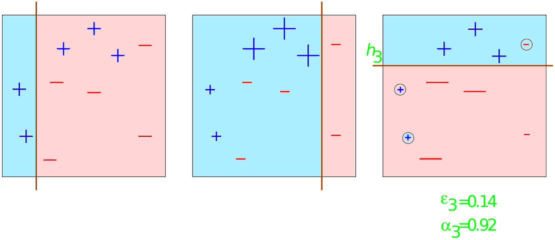
(Classical) Example (Freund & Schapire)

Iteration 2:



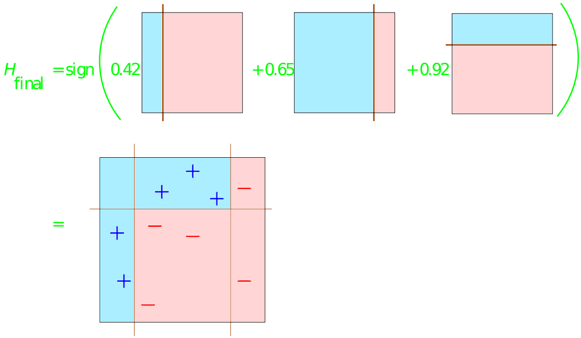
(Classical) Example (Freund & Schapire)

Iteration 3:



(Classical) Example (Freund & Schapire)

Final classifier:



Training error theorem: let $\epsilon_t < 1/2$ be the error rate at step t and let $\gamma_t = 1/2 - \epsilon_t$, then the training error of the final classifier is upper bounded by

$$\mathrm{Err}_{\mathrm{train}}(H) \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right)$$

- then if $\forall t : \gamma_t \geq \gamma > 0$, $\mathrm{Err}_{\mathrm{train}} \leq \exp(-2\gamma^2 T)$
- it follows that $\mathrm{Err}_{\mathrm{train}} \to 0$ as $T \to \infty$
- if $\gamma_t \gg \gamma$ the convergence is much faster

What about overfitting?

- Occam's razor suggests that simpler rules are preferable
- for SVMs, sparser models (less SVs) have better generalization properties
- AdaBoost?

What about overfitting?

- Occam's razor suggests that simpler rules are preferable
- for SVMs, sparser models (less SVs) have better generalization properties
- AdaBoost?
- practice shows that AdaBoost is resistant to overfitting, in normal conditions

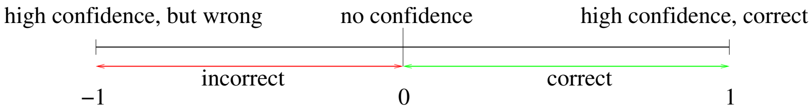
What about overfitting?

- Occam's razor suggests that simpler rules are preferable
- for SVMs, sparser models (less SVs) have better generalization properties
- AdaBoost?
- practice shows that AdaBoost is resistant to overfitting, in normal conditions
- in highly noisy conditions, AdaBoost can overfit! Regularized versions exist to tackle this situation

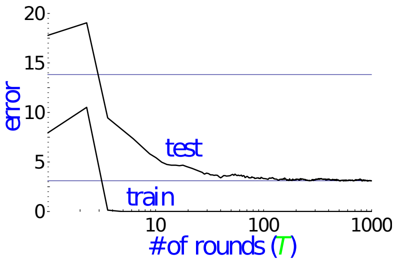
Where does the robustness (to overfitting) come from?

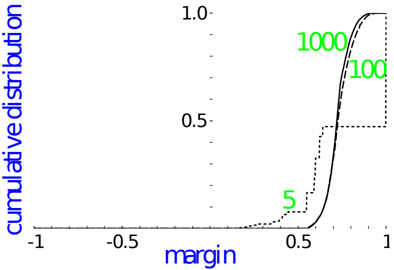
- it's a matter of margin! (most likely)
- define the margin as the "strength of the vote", i.e. "weighted fraction of correct votes" - "weighted fraction of incorrect votes"

The output from the final classifier (before $\text{sign}()$) $\in [-1, 1]$:



Example (from F&S's tutorial): the "letters" data set from UCI, C4.5 weak classifiers



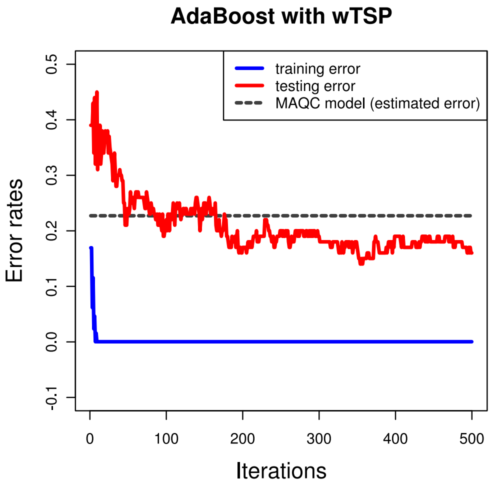


	#rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

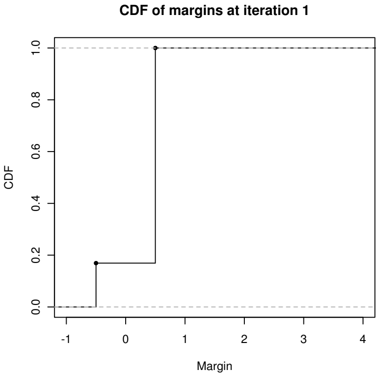
...and a real world example: prediction of pCR in breast cancer

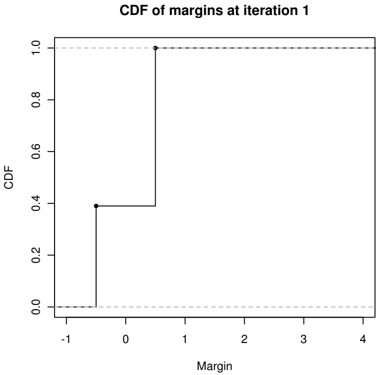
- AdaBoost with weighted top scoring pairs weak classifiers
- data: MDA gene expression data ($\sim$22,000 variables) from MAQC project: $n = 130$ training samples, $n = 100$ testing samples
- data comes different hospitals, clinical series, no much control on the representativeness of the training set
- endpoint: pathologic complete response (pCR)

Training and testing errors (with functional margin)

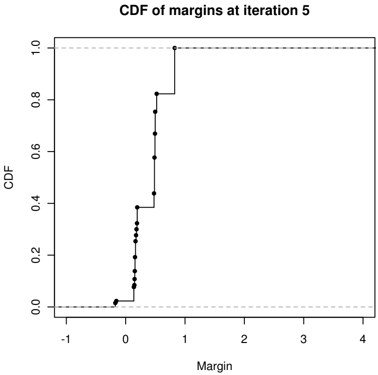


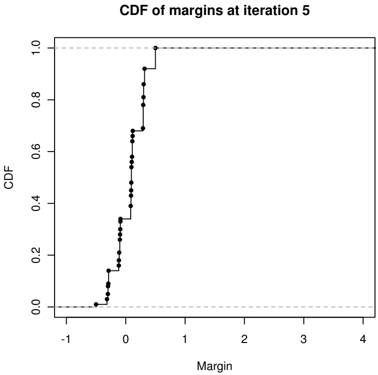
iterations = 1, $err_{tr} = 0.17, err_{ts} = 0.39$



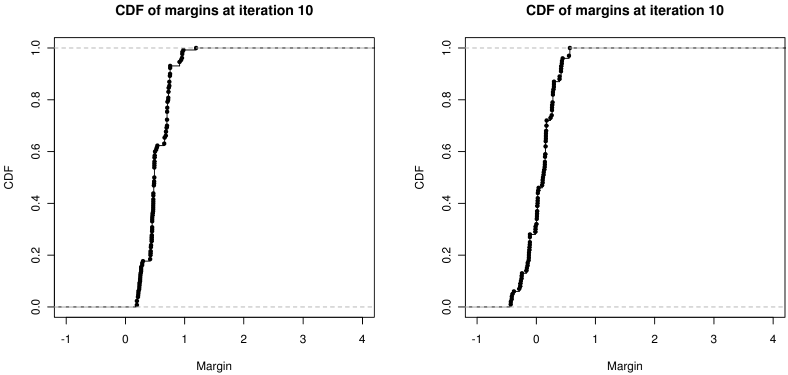


iterations = 5, $err_{tr} = 0.02$, $err_{ts} = 0.34$

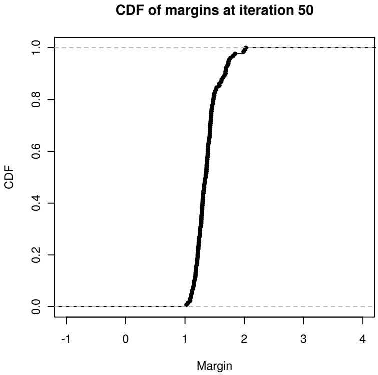


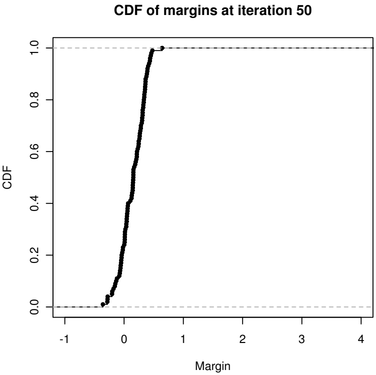


iterations = 10, $err_{tr} = 0.0$, $err_{ts} = 0.31$

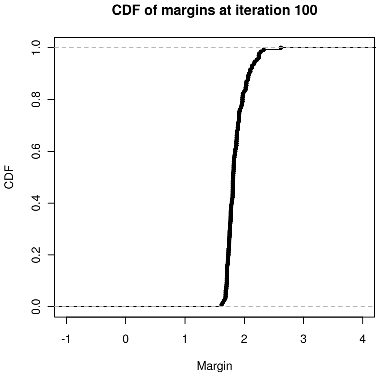


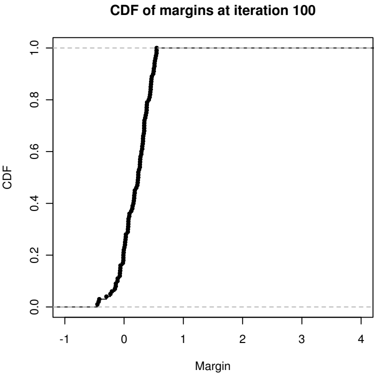
iterations = 50, $err_{tr} = 0.0$, $err_{ts} = 0.23$





iterations = 100, $err_{tr} = 0.0$, $err_{ts} = 0.22$





Ideas:

- large margin allows a sparser approximation of the final classifier, hence the final classifier should have better generalization properties than its size would suggest
- the AdaBoost increases the margin as T grows and decreases the effective complexity of the final classifier
- $\forall \theta > 0$, $\text{Err}(H) \leq \hat{\Pr}[\text{margin} \leq \theta] + \mathcal{O}(\sqrt{h/n}/\theta)$ where h is the "complexity" of weak classifiers
- $\hat{\Pr}[\text{margin} \leq \theta] \rightarrow 0$ exponentially fast in T if $\gamma_t > \theta$

Outline

A few different interpretations

- game theory: AdaBoost classifier as a solution of a minmax game
- loss minimization
- additive logistic model
- maximum entropy
- etc. etc.

AdaBoost as a minimizer of exponential loss

- let $L(y, f(\mathbf{x}))$ be the *loss function* measuring the discrepancies between true target (label or real value) y and the predicted value $f(\mathbf{x})$
- it can be shown that AdaBoost minimizes (remember the scaling factor Z_t?)

$$\prod_t Z_t = \frac{1}{n} \sum_i \exp(-y_i f(\mathbf{x}_i))$$

  where $f(\mathbf{x}) = \sum_t \alpha_t h_t(\mathbf{x})$
- $yf(\mathbf{x})$ is the (functional) margin, similar to SVM
- exponential loss is an upper bound of the 0-1 loss
- AdaBoost is a greedy procedure for loss minimization: α_t and h_t are chosen locally to minimize the current loss

Coordinate descent [Breiman]

- let $\{h_1, \ldots, h_m\}$ be the space of all weak classifiers
- the goal is to find $\beta_1, \ldots, \beta_m$ (coordinates in the space of weak classifiers) where the loss

$$L(\beta_1, \ldots, \beta_m) = \sum_i \exp(-y_i \sum_k \beta_k h(\mathbf{x}_i))$$

  is minimized
- coordinate descent procedure:
  - start with $\beta_k = 0$
  - at each step: choose coordinate β_k (on axis h_t) and update it by an increment α_t
  - α_t is chosen to maximize the decrease in loss
- this is the very procedure implemented by AdaBoost

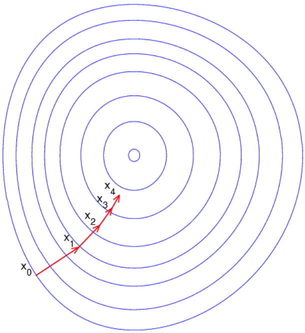
Outline

Gradient descent optimization (reminder)

- let Ω be a differentiable optimization criterion
- let $x_k = x_{k-1} - \gamma \nabla \Omega(x_{k-1})$, then for some small $\gamma > 0$ $\Omega(x_k) \leq \Omega(x_{k-1})$

Issues:

- slow convergence
- sensitive to initial point



Gradient descent in function space

- in the following, we will generalize from ± 1-valued classifiers to real-valued functions
- change of notation: F becomes the generalized version of H and f the generalized version of h, respectively
- $F_M(\mathbf{x}) = \sum_1^M f_m(\mathbf{x})$ is evaluated at each $\mathbf{x}$
- *gradient (steepest) descent*:

$$f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x}) = -\rho_m \nabla_F \left[\mathrm{E}_{y,\mathbf{x}} \left[L(y, F(\mathbf{x})) \right] \right]_{F=F_{m-1}}$$

$$\rho_m = \arg\min_{\rho} \mathrm{E}_{y,\mathbf{x}} \left[L(y, F_{m-1}(\mathbf{x}) - \rho g_m(\mathbf{x})) \right]$$

Additive models

Friedman, Hastie, Tibshirani, *Additive logistic regression: a statistical view of boosting*, The Annals of Statistics, 2000.

- regression models: let $y \in \mathbb{R}$ and model the mean:

$$\mathrm{E}\left[y|\mathbf{x}\right] = \sum_{j=1}^{p} f_j(x_j),$$

where $\mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p$.

- iteratively update (backfit) the current approximation until convergence:

$$f_j(x_j) \leftarrow \mathrm{E}\left[y - \sum_{k \neq j} f_k(x_k) \,\middle|\, x_j \right].$$

- the final solution, $F(\mathbf{x}) = \sum_1^p f_j(x_j)$, is a minimizer of

$$\mathrm{E}\left[(y - F(\mathbf{x}))^2 \right].$$

Extended additive models

- consider a family of functions

$$f_m(\mathbf{x}) = \beta_m b(\mathbf{x}; \gamma_m).$$

- $b(\cdot)$: basis functions (linear, sigmoid, RBF, wavelets,. . .)
- Notes on basis functions:
  - span a function subspace
  - they need not be orthogonal, nor form a complete/minimal base
  - they can be chosen to form a *redundant dictionary*: matching pursuit
- applications in (statistical) signal processing; image compression; multi–scale data analysis;...

Fitting the model:

- generalized backfitting:

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta,\gamma} \mathrm{E}\left[\left(y - (\sum_{k \neq m} \beta_k b(\mathbf{x}; \gamma_k) + \beta b(\mathbf{x}; \gamma))\right)^2\right]$$

- greedy optimization: let $F_M(\mathbf{x}) = \sum_1^M \beta_m b(\mathbf{x}; \gamma_m)$ be the solution after M iterations; the successive approximations are

$$\{\beta_m, \gamma_m\} = \arg\min_{\beta,\gamma} \mathrm{E}\left[(y - (F_{m-1}(\mathbf{x}) + \beta b(\mathbf{x}; \gamma))^2\right]$$

$\rightarrow$ matching pursuit; in classification: kernel matching pursuit

Mallat, Zhang, *Matching pursuit with time–frequency dictionaries*, 1993
Vincent, Bengio, *Kernel matching pursuit*, 2002
Popovici, Thiran, *Kernel matching pursuit for large datasets*, 2005

From regression to classification

- goal (for binary problems): estimate $\Pr(y = 1|\mathbf{x})$
- logistic regression:

$$\ln \frac{\Pr(y = 1|\mathbf{x})}{\Pr(y = -1|\mathbf{x})} = F_M(\mathbf{x})$$

with $F_M(\mathbf{x}) \in \mathbb{R}$.

- $\Leftrightarrow p(\mathbf{x}) = \Pr(y = 1|\mathbf{x}) = \frac{\exp(F_M(\mathbf{x}))}{1+\exp(F_M(\mathbf{x}))}$
- F_M is obtained by minimizing the *expected loss*:

$$F_M(\mathbf{x}) = \arg\min_F \mathrm{E}_{y,\mathbf{x}}\left[L(y, F(\mathbf{x}))\right] = \arg\min_F \mathrm{E}_{\mathbf{x}}\left[\mathrm{E}_y\left[L(y, F(\mathbf{x}))\right] |\mathbf{x}\right]$$

Generalized boosting algorithm

given $\{(\mathbf{x}_i, y_i) | i = 1, \ldots, N\}$, let $F_0(\mathbf{x}) = f_0(\mathbf{x})$
for all $m = 1, \ldots, M$ **do**
    compute the current negative gradient:

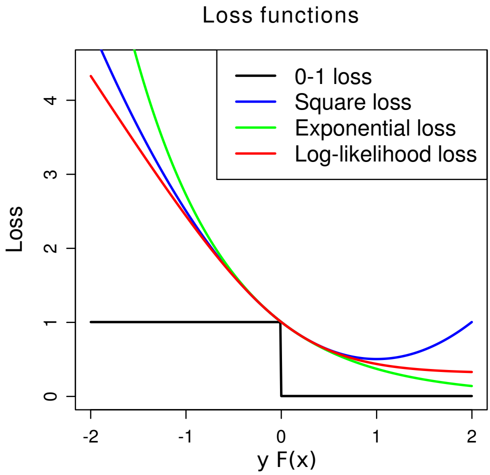
$$z_i = -\nabla_F L(F)\Big|_{F=F_{m-1}} = -\left.\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F}\right|_{F=F_{m-1}(\mathbf{x}_i)}$$

    and fit f_m using the new set $\{(\mathbf{x}_i, z_i) | i = 1, \ldots, N\}$
    find the step–size

$$c_m = \arg\min_c \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + c f_m(\mathbf{x}_i))$$

    let $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + c_m f_m(\mathbf{x})$
end for
return final classifier $\text{sign}\,[F_M(\mathbf{x})]$

Which loss function?



Exponential loss

$$L(y, F) = \mathrm{E}\left[e^{-yF(\mathbf{x})}\right]$$

Notes:

- $L(y, F)$ is minimized at

$$F(\mathbf{x}) = \frac{1}{2} \ln \frac{\Pr(y = 1|\mathbf{x})}{\Pr(y = -1|\mathbf{x})}$$

- $\frac{yF(\mathbf{x})}{\|F\|}$ is called *margin of sample* $\mathbf{x} \Rightarrow L(y, F)$ forces margin maximization
- L is differentiable and an upper bound of $\mathbf{1}_{[yF(\mathbf{x})<0]}$
- L has the same population minimizer as the binomial log–likelihood

AdaBoost builds an additive logistic regression model

let $w_i = 1/N$
for all $m = 1, \ldots, M$ **do**
  fit the *weak classifier* $f_m(\mathbf{x}) \in \{\pm 1\}$ using the weights w_i on the training data
  $err_m = \mathrm{E}_w \left[\mathbf{1}_{[y \neq f_m(\mathbf{x})]} \right]$ { expectation with respect to weights! }
  $c_m = \ln \frac{1-err_m}{err_m}$ (note: $c_m = 2 \arg\min_c L(\sum_1^{m-1} f_i + c f_m)$)
  update the weights

$$w_i \leftarrow w_i \exp\left(c_m \mathbf{1}_{[y_i \neq f_m(\mathbf{x}_i)]}\right), \; i = 1, \ldots, N$$

  and normalize such that $||w|| = 1$
end for
return final classifier $\mathrm{sign}\left[\sum_{m=1}^{M} c_m f_m(\mathbf{x})\right]$

Real AdaBoost: stagewise optimization of exponential loss

let $w_i = 1/N$
for all $m = 1, \ldots, M$ **do**
    fit the *weak classifier* using the weights w_i on the training data and obtain the posteriors

$$p_m(\mathbf{x}) = \hat{P}_w(y = 1|\mathbf{x}) \in [0, 1]$$

    let $f_m(\mathbf{x}) = \frac{1}{2} \ln \frac{p_m(\mathbf{x})}{1-p_m(\mathbf{x})}$ {note: this is the local minimizer of L}
    update the weights

$$w_i \leftarrow w_i \exp\left(-y_i f_m(\mathbf{x}_i)\right), \; i = 1, \ldots, N$$

    and normalize such that $||w|| = 1$
end for
return final classifier $\text{sign}\left[\sum_{m=1}^{M} f_m(\mathbf{x})\right]$

LogitBoost: stagewise opt. of binomial log–likelihood

Let $y^* = (1+y)/2 \in \{0,1\}$ and
$\Pr(y^* = 1|\mathbf{x}) = p = \exp(F(\mathbf{x}))/(\exp(F(\mathbf{x})) + \exp(-F(\mathbf{x})))$

1: let $w_i = 1/N, p_i = 1/2, \forall i = 1, \ldots, N,\ F(\mathbf{x}) = 0$
2: **for all** $m = 1, \ldots, M$ **do**
3: let $z_i = \frac{y_i^* - p_i}{p_i(1-p_i)}$ { new responses, instead of y}
4: let $w_i = p_i(1 - p_i)$
5: fit f_m by weighted least–square regression of z_i to $\mathbf{x}_i$ using weights w_i
6: update $F \leftarrow F + 1/2 f_m$
7: update $p \leftarrow \exp(F)/(\exp(F) + \exp(-F))$
8: **end for**
9: **return** final classifier $\text{sign}\left[\sum_{m=1}^{M} f_m(\mathbf{x})\right]$

Which weak learner?

- any classifier with an error rate < 0.5
- decision stumps (classification tree with 1 node)
- classical classification trees
- top scoring pairs classifier
- linear (logistic) regression (an example later)
- radial basis functions
- ...

Practical issues

- the weak classifier should not be too strong
- AdaBoost or LogitBoost are good first choices for classification problems
- stopping rules:
  - quit when the weak classifier cannot fit the data anymore
  - choose M by an inner cross–validation or independent data set
  - use AIC, BIC, MDL as criteria for choosing M