# PV211: Introduction to Information Retrieval

https://www.fi.muni.cz/~sojka/PV211

## IIR 12: Language Models for IR
## Handout version

Petr Sojka, Hinrich Schütze et al.

Faculty of Informatics, Masaryk University, Brno
Center for Information and Language Processing, University of Munich

2023-04-12

(compiled on 2023-04-13 20:25:56)

# Overview

# Take-away today

- What matters in language modeling? Feature (term) selection for text classification and similarity
- Statistical language models: Introduction
- Statistical language models in IR
- Large language models
- Discussion: Properties of different probabilistic models in use in IR, hype of LLM

# Language modeling (in IR)

A language model is a probability distribution over sequences of words.

Language models are used in IR retrieval in the query likelihood model. There, a separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query $Q$ in the document's language model $M_d : P(Q \mid M_d)$.

Commonly, the unigram language model (bag of words model) is used for this purpose.

An $n$-gram language model is a language model that models sequences of words as a Markov process.

Subword (character (Morse, Turing), syllable) or phrase models (SWOMPT in English) are also possible.

# Latest milestones in language modeling

- 2013: Mikolov et al.: Efficient Estimation of Word Representations in Vector Space (word2vec)
- 2017: Vaswani et al.: Attention is all you need
- 2018: Generative pretrained models (GPT): large language models consisting of deep neural networks with billions of trainable parameters, trained on massive datasets of unlabelled text, have demonstrated impressive results on a wide variety of natural language processing tasks. This development has led to a shift in research focus toward the use of general-purpose LLMs.

# Which language model is best?

- Perplexity: In information theory, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. A low perplexity indicates the probability distribution is good at predicting the sample.
- https://en.wikipedia.org/wiki/Perplexity
- https://huggingface.co/docs/transformers/perplexity

# Feature (term) selection

- In text classification, we usually represent documents in a high-dimensional space, with each dimension corresponding to a term.
- In this lecture: axis = dimension = word = term = feature
- Many dimensions correspond to rare words.
- Rare words can mislead the classifier.
- Rare misleading features are called noise features.
- Eliminating noise features from the representation increases efficiency and effectiveness of text classification.
- Eliminating features is called feature selection.

# Example for a noise feature

- Let's say we're doing text classification for the class *China*.
- Suppose a rare term, say ARACHNOCENTRIC, has no information about *China* . . .
- . . . but all instances of ARACHNOCENTRIC happen to occur in *China* documents in our training set.
- Then we may learn a classifier that incorrectly interprets ARACHNOCENTRIC as evidence for the class *China*.
- Such an incorrect generalization from an accidental property of the training set is called overfitting.
- Feature selection reduces overfitting and improves the accuracy of the classifier.

# Basic feature selection algorithm

$\textsc{SelectFeatures}(\mathbb{D}, c, k)$
$V \leftarrow \textsc{ExtractVocabulary}(\mathbb{D})$
$L \leftarrow []$
**for each** $t \in V$
**do** $A(t, c) \leftarrow \textsc{ComputeFeatureUtility}(\mathbb{D}, t, c)$
    $\textsc{Append}(L, \langle A(t, c), t \rangle)$
**return** $\textsc{FeaturesWithLargestValues}(L, k)$

How do we compute $A$, the feature utility?

# Different feature selection methods

- A feature selection method is mainly defined by the feature utility measure it employs
- Feature utility measures:
  - Frequency – select the most frequent terms
  - Mutual information – select the terms with the highest mutual information
  - Mutual information is also called information gain in this context.
  - Chi-square (see book)

# Mutual information

- Compute the feature utility $A(t, c)$ as the mutual information (MI) of term $t$ and class $c$.
- MI tells us "how much information" the term contains about the class and vice versa.
- For example, if a term's occurrence is independent of the class (same proportion of docs within/without class contain the term), then MI is 0.
- Definition:

$$I(U; C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t) P(C = e_c)}$$

# How to compute MI values

- Based on maximum likelihood estimates, the formula we actually use is:

$$\begin{aligned} I(U;C) &= \frac{N_{11}}{N}\log_2\frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N}\log_2\frac{NN_{01}}{N_{0.}N_{.1}} \\ &\quad + \frac{N_{10}}{N}\log_2\frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N}\log_2\frac{NN_{00}}{N_{0.}N_{.0}} \end{aligned}$$

- $N_{10}$: number of documents that contain $t$ ($e_t = 1$) and are not in $c$ ($e_c = 0$); $N_{11}$: number of documents that contain $t$ ($e_t = 1$) and are in $c$ ($e_c = 1$); $N_{01}$: number of documents that do not contain $t$ ($e_t = 1$) and are in $c$ ($e_c = 1$); $N_{00}$: number of documents that do not contain $t$ ($e_t = 1$) and are not in $c$ ($e_c = 1$); $N = N_{00} + N_{01} + N_{10} + N_{11}$.

# How to compute MI values (2)

- Alternative way of computing MI:

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U=e_t, C=e_c) \log_2 \frac{N(U=e_t, C=e_c)}{E(U=e_t)E(C=e_c)}$$

- $N(U=e_t, C=e_c)$ is the count of documents with values $e_t$ and $e_c$ .
- $E(U=e_t, C=e_c)$ is the expected count of documents with values $e_t$ and $e_c$ if we assume that the two random variables are independent.

# MI example for *poultry*/EXPORT in Reuters

| | $e_c = e_{poultry} = 1$ | $e_c = e_{poultry} = 0$ |
|---|---|---|
| $e_t = e_{\text{EXPORT}} = 1$ | $N_{11} = 49$ | $N_{10} = 27{,}652$ |
| $e_t = e_{\text{EXPORT}} = 0$ | $N_{01} = 141$ | $N_{00} = 774{,}106$ |

Plug these values into formula:

$$
\begin{aligned}
I(U;C) &= \frac{49}{801{,}948}\log_2\frac{801{,}948\cdot 49}{(49+27{,}652)(49+141)}\\
&\quad+\frac{141}{801{,}948}\log_2\frac{801{,}948\cdot 141}{(141+774{,}106)(49+141)}\\
&\quad+\frac{27{,}652}{801{,}948}\log_2\frac{801{,}948\cdot 27{,}652}{(49+27{,}652)(27{,}652+774{,}106)}\\
&\quad+\frac{774{,}106}{801{,}948}\log_2\frac{801{,}948\cdot 774{,}106}{(141+774{,}106)(27{,}652+774{,}106)}\\
&\approx 0.000105
\end{aligned}
$$

# MI feature selection on Reuters

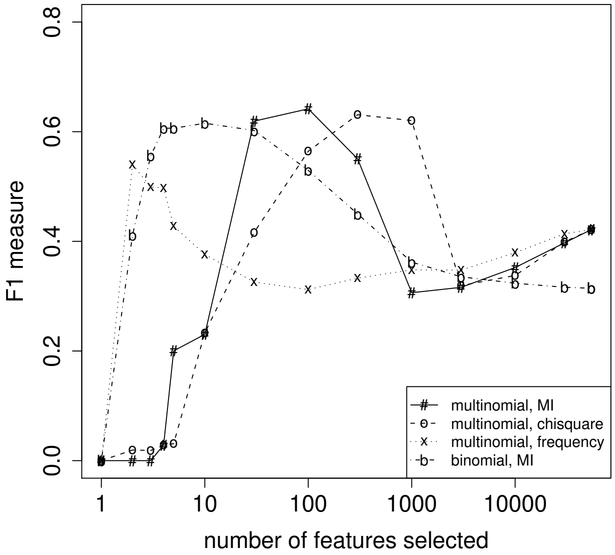
Class: *coffee*

| term | MI |
|---|---|
| COFFEE | 0.0111 |
| BAGS | 0.0042 |
| GROWERS | 0.0025 |
| KG | 0.0019 |
| COLOMBIA | 0.0018 |
| BRAZIL | 0.0016 |
| EXPORT | 0.0014 |
| EXPORTERS | 0.0013 |
| EXPORTS | 0.0013 |
| CROP | 0.0012 |

Class: *sports*

| term | MI |
|---|---|
| SOCCER | 0.0681 |
| CUP | 0.0515 |
| MATCH | 0.0441 |
| MATCHES | 0.0408 |
| PLAYED | 0.0388 |
| LEAGUE | 0.0386 |
| BEAT | 0.0301 |
| GAME | 0.0299 |
| GAMES | 0.0284 |
| TEAM | 0.0264 |

# Naive Bayes: Effect of feature selection




(multinomial = multinomial Naive Bayes, binomial = Bernoulli Naive Bayes)

# Feature selection for Naive Bayes

- In general, feature selection is necessary for Naive Bayes to get decent performance.
- Also true for many other learning methods in text classification: you need feature selection for optimal performance.

# Exercise

(i) Compute the "export"/POULTRY contingency table for the "Kyoto"/JAPAN in the collection given below. (ii) Make up a contingency table for which MI is 0 – that is, term and class are independent of each other.

"export"/POULTRY table:

| | $e_c = e_{poultry} = 1$ | $e_c = e_{poultry} = 0$ |
|---|---|---|
| $e_t = e_{\text{EXPORT}} = 1$ | $N_{11} = 49$ | $N_{10} = 27{,}652$ |
| $e_t = e_{\text{EXPORT}} = 0$ | $N_{01} = 141$ | $N_{00} = 774{,}106$ |

Collection:

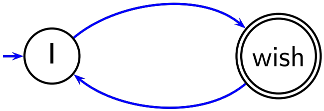
| | docID | words in document | in $c$ = Japan? |
|---|---|---|---|
| training set | 1 | Kyoto Osaka Taiwan | yes |
| | 2 | Japan Kyoto | yes |
| | 3 | Taipei Taiwan | no |
| | 4 | Macao Taiwan Shanghai | no |
| | 5 | London | no |

# Using language models (LMs) for IR

1. LM = language model
2. We view the document as a generative model that generates the query.
3. What we need to do:
4. Define the precise generative model we want to use
5. Estimate parameters (different parameters for each document's model)
6. Smooth to avoid zeros
7. Apply to query and find document most likely to have generated the query
8. Present most likely document(s) to user
9. Note that 4–7 is very similar to what we did in Naive Bayes.

# What is a language model?

We can view a finite state automaton as a deterministic language model.

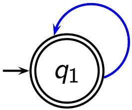



I wish I wish I wish I wish . . .

Cannot generate: "wish I wish" or "I wish I"

Our basic model: each document was generated by a different automaton like this except that these automata are probabilistic.

# A probabilistic language model




| $w$ | $P(w\|q_1)$ | $w$ | $P(w\|q_1)$ |
|---|---|---|---|
| STOP | 0.2 | toad | 0.01 |
| the | 0.2 | said | 0.03 |
| a | 0.1 | likes | 0.02 |
| frog | 0.01 | that | 0.04 |
| | | ... | ... |

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state $q_1$.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048$
0.01* 0.03* 0.04* 0.01* 0.02* 0.01* 0.2

# A different language model for each document

language model of $d_1$

| $w$ | $P(w\|.)$ | $w$ | $P(w\|.)$ |
|---|---|---|---|
| STOP | .2 | toad | .01 |
| the | .2 | said | .03 |
| a | .1 | likes | .02 |
| frog | .01 | that | .04 |
| | | ... | ... |

language model of $d_2$

| $w$ | $P(w\|.)$ | $w$ | $P(w\|.)$ |
|---|---|---|---|
| STOP | .2 | toad | .02 |
| the | .15 | said | .03 |
| a | .08 | likes | .02 |
| frog | .01 | that | .05 |
| | | ... | ... |

query: frog said that toad likes frog STOP

$P(\text{query}|M_{d1}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048 = 4.8 \cdot 10^{-12}$

$P(\text{query}|M_{d2}) = 0.01 \cdot 0.03 \cdot 0.05 \cdot 0.02 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000120 = 12 \cdot 10^{-12}$
0.01* 0.03* 0.05* 0.02* 0.02* 0.01* 0.2

$P(\text{query}|M_{d1}) < P(\text{query}|M_{d2})$ Thus, document $d_2$ is "more relevant" to the query "frog said that toad likes frog STOP" than

# Using language models in IR

- Each document is treated as (the basis for) a language model.
- Given a query $q$
- Rank documents based on $P(d|q)$
- $$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$
- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all $d$
  - But we can give a higher prior to "high-quality" documents, e.g., those with high PageRank.
- $P(q|d)$ is the probability of $q$ given $d$ .
- For uniform prior: ranking documents according according to $P(q|d)$ and $P(d|q)$ is equivalent.

# Where we are

- In the LM approach to IR, we attempt to model the query generation process.
- Then we rank documents by the probability that a query would be observed as a random sample from the respective document model.
- That is, we rank according to $P(q|d)$.
- Next: how do we compute $P(q|d)$?

# How to compute $P(q|d)$

- We will make the same conditional independence assumption as for Naive Bayes.
-

$$P(q|M_d) = P(\langle t_1, \dots, t_{|q|} \rangle | M_d) = \prod_{1 \leq k \leq |q|} P(t_k|M_d)$$

($|q|$: length of $q$; $t_k$: the token occurring at position $k$ in $q$)

- This is equivalent to:

$$P(q|M_d) = \prod_{\text{distinct term } t \text{ in } q} P(t|M_d)^{\mathrm{tf}_{t,q}}$$

- $\mathrm{tf}_{t,q}$: term frequency (# occurrences) of $t$ in $q$
- Multinomial model (omitting constant factor)

# Parameter estimation

- Missing piece: Where do the parameters $P(t|M_d)$ come from?
- Start with maximum likelihood estimates (as we did for Naive Bayes)
- $$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

  ($|d|$: length of $d$; $\mathrm{tf}_{t,d}$: # occurrences of $t$ in $d$)
- As in Naive Bayes, we have a problem with zeros.
- A single $t$ with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- We would give a single term "veto power".
- For example, for query [Michael Jackson top hits] a document about "top songs" (but not using the word "hits") would have $P(q|M_d) = 0$. – Thats's bad.
- We need to smooth the estimates to avoid zeros.

# Smoothing

- Key intuition: A nonoccurring term is possible (even though it didn't occur), ...
- ...but no more likely than would be expected by chance in the collection.
- Notation: $M_c$: the collection model; $\mathrm{cf}_t$: the number of occurrences of $t$ in the collection; $T = \sum_t \mathrm{cf}_t$: the total number of tokens in the collection.
- $$\hat{P}(t|M_c) = \frac{\mathrm{cf}_t}{T}$$
- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

# Jelinek-Mercer smoothing

- $P(t|d) = \lambda P(t|M_d) + (1 - \lambda)P(t|M_c)$
- Mixes the probability from the document with the general collection frequency of the word.
- High value of $\lambda$: "conjunctive-like" search – tends to retrieve documents containing all query words.
- Low value of $\lambda$: more disjunctive, suitable for long queries
- Correctly setting $\lambda$ is very important for good performance.

# Jelinek-Mercer smoothing: Summary

- $$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1-\lambda)P(t_k|M_c))$$
- What we model: The user has a document in mind and generates the query from this document.
- The equation represents the probability that the document that the user had in mind was in fact this one.

# Example

- Collection: $d_1$ and $d_2$
- $d_1$: Jackson was one of the most talented entertainers of all time
- $d_2$: Michael Jackson anointed himself King of Pop
- Query $q$: Michael Jackson
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(0/11 + 1/18)/2] \cdot [(1/11 + 2/18)/2] \approx 0.003$
- $P(q|d_2) = [(1/7 + 1/18)/2] \cdot [(1/7 + 2/18)/2] \approx 0.013$
- Ranking: $d_2 > d_1$

# Exercise: Compute ranking

- Collection: $d_1$ and $d_2$
- $d_1$: Xerox reports a profit but revenue is down
- $d_2$: Lucene narrows quarter loss but revenue decreases further
- Query $q$: revenue down
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(1/8 + 2/16)/2] \cdot [(1/8 + 1/16)/2] = 1/8 \cdot 3/32 = 3/256$
- $P(q|d_2) = [(1/8 + 2/16)/2] \cdot [(0/8 + 1/16)/2] = 1/8 \cdot 1/32 = 1/256$
- Ranking: $d_1 > d_2$

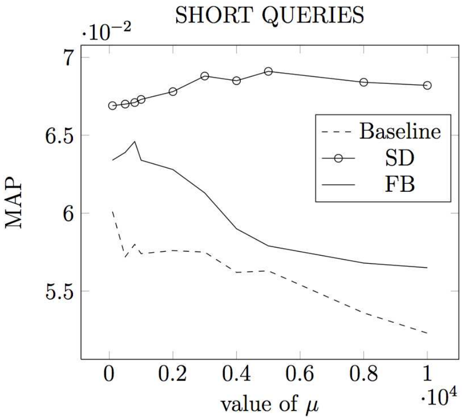
# Dirichlet smoothing

- $$\hat{P}(t|d) = \frac{\mathrm{tf}_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha}$$
- The background distribution $\hat{P}(t|M_c)$ is the prior for $\hat{P}(t|d)$.
- Intuition: Before having seen any part of the document we start with the background distribution as our estimate.
- As we read the document and count terms we update the background distribution.
- The weighting factor $\alpha$ determines how strong an effect the prior has.

# Jelinek-Mercer or Dirichlet?

- Dirichlet performs better for keyword queries, Jelinek-Mercer performs better for verbose queries.
- Both models are sensitive to the smoothing parameters – you shouldn't use these models without parameter tuning.

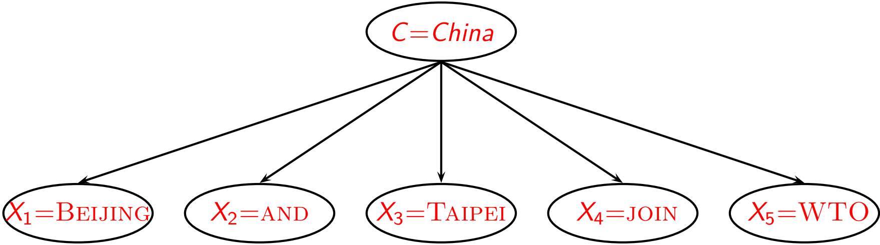
# Sensitivity of Dirichlet to smoothing parameter




$\mu$ is the Dirichlet smoothing parameter (called $\alpha$ on the previous slides)

# Language models are generative models

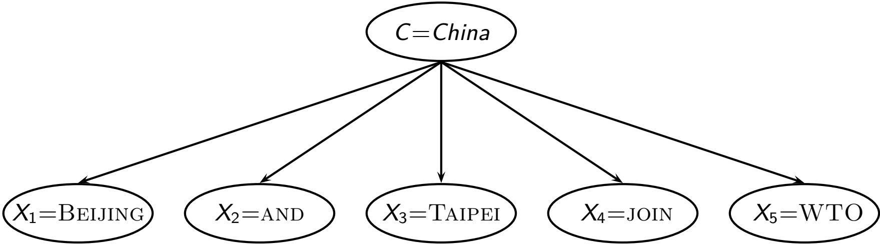
We have assumed that queries are generated by a probabilistic process that looks like this: (as in Naive Bayes)



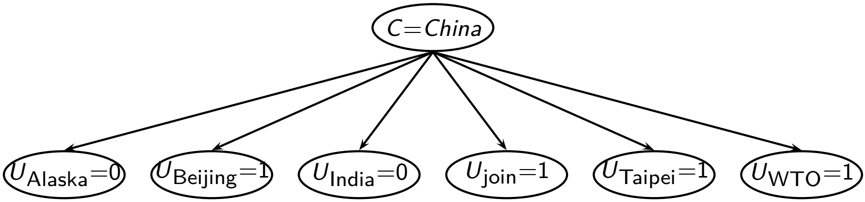
# Naive Bayes and LM generative models

- We want to classify document $d$.
  We want to classify a query $q$.
  - Classes: e.g., geographical regions like *China*, *UK*, *Kenya*.
    Each document in the collection is a different class.
- Assume that $d$ was generated by the generative model.
  Assume that $q$ was generated by a generative model
- Key question: Which of the classes is most likely to have generated the document? Which document (=class) is most likely to have generated the query $q$?
  - Or: for which class do we have the most evidence? For which document (as the source of the query) do we have the most evidence?

# Naive Bayes Multinomial model / IR language models



# Naive Bayes Bernoulli model / Binary independence model



# Comparison of the two models

| | multinomial model / IR language model | Bernoulli model / BIM |
|---|---|---|
| event model | generation of (multi)set of tokens | generation of subset of voc |
| random variable(s) | $X = t$ iff $t$ occurs at given pos | $U_t = 1$ iff $t$ occurs in doc |
| doc. representation | $d = \langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle, t_k \in V$ | $d = \langle e_1, \ldots, e_i, \ldots, e_M \rangle$, $e_i \in \{0, 1\}$ |
| parameter estimation | $\hat{P}(X = t \mid c)$ | $\hat{P}(U_i = e \mid c)$ |
| dec. rule: maximize | $\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k \mid c)$ | $\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i \mid c)$ |
| multiple occurrences | taken into account | ignored |
| length of docs | can handle longer docs | works best for short docs |
| # features | can handle more | works best with fewer |
| estimate for THE | $\hat{P}(X = \text{the} \mid c) \approx 0.05$ | $\hat{P}(U_{\text{the}} = 1 \mid c) \approx 1.0$ |

# LMs vs. Multinomial Naive Bayes

- We classify the query in LMs; we classify documents in text classification.
- Each document is a class in LMs vs. classes are human-defined in text classification

# Vector space (tf-idf) vs. LM

| | precision | | | significant |
|---|---|---|---|---|
| Rec. | tf-idf | LM | %chg | |
| 0.0 | 0.7439 | 0.7590 | +2.0 | |
| 0.1 | 0.4521 | 0.4910 | +8.6 | |
| 0.2 | 0.3514 | 0.4045 | +15.1 | * |
| 0.4 | 0.2093 | 0.2572 | +22.9 | * |
| 0.6 | 0.1024 | 0.1405 | +37.1 | * |
| 0.8 | 0.0160 | 0.0432 | +169.6 | * |
| 1.0 | 0.0028 | 0.0050 | +76.9 | |
| 11-point average | 0.1868 | 0.2233 | +19.6 | * |

The language modeling approach *always does better* in these experiments . . .

. . . but note that where the approach shows significant gains is at higher levels of recall.

# Vector space vs BM25 vs LM

- BM25/LM: based on probability theory
- Vector space: based on similarity, a geometric/linear algebra notion
- Term frequency is directly used in all three models.
  - LMs: raw term frequency, BM25/Vector space: more complex
- Length normalization
  - Vector space: Cosine or pivot normalization
  - LMs: probabilities are inherently length normalized
  - BM25: tuning parameters for optimizing length normalization
- idf: BM25/vector space use it directly.
- LMs: Mixing term and collection frequencies has an effect similar to idf.
  - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.
- Collection frequency (LMs) vs. document frequency (BM25, vector space)

# Language models for IR: Assumptions

- Simplifying assumption: Queries and documents are objects of the same type. Not true!
  - There are other LMs for IR that do not make this assumption.
  - The vector space model makes the same assumption.
- Simplifying assumption: Terms are conditionally independent.
  - Again, vector space model (and Naive Bayes) make the same assumption.
- Cleaner statement of assumptions than vector space
- Thus, better theoretical foundation than vector space
  - . . . but “pure” LMs perform much worse than “tuned” LMs.

# Take-away today

- What matters in language modeling? Feature (term) selection for text classification and similarity
- Statistical language models: Introduction
- Statistical language models in IR
- Large language models
- Discussion: Properties of different probabilistic models in use in IR, hype of LLM

# Resources

- Chapter 13 of IIR (feature selection)
- Chapter 12 of IIR
- Resources at `https://www.fi.muni.cz/~sojka/PV211/` and `http://cislmu.org`, materials in MU IS and FI MU library
  - Ponte and Croft's 1998 SIGIR paper (one of the first on LMs in IR)
  - Zhai and Lafferty: A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst. (2004).
  - Lemur toolkit (good support for LMs in IR)