NPFL103: Information Retrieval (8)

Language Models for Information Retrieval, Text Classification

Pavel Pecina

pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Original slides are courtesy of Hinrich Schütze, University of Stuttgart.

Contents

Language models

Using language models for Information Retrieval

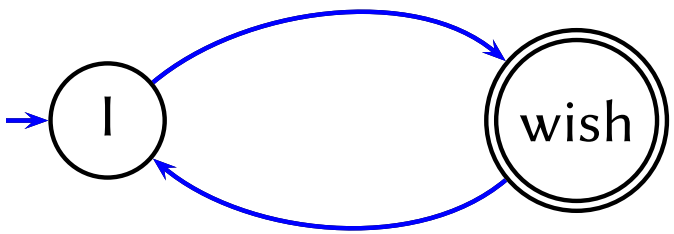
View the document d as a generative model that generates the query q.

What we need to do:

1. Define the precise generative model we want to use
2. Estimate parameters (different for each document's model)
3. Smooth to avoid zeros
4. Apply to query and find document most likely to generate the query
5. Present most likely document(s) to user

What is a language model?

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

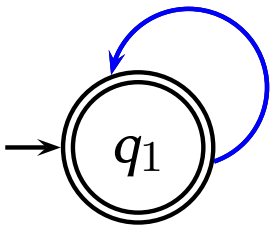


I wish I wish I wish I wish ...

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 a special symbol indicating that the automaton stops.

Example: frog said that toad likes frog STOP

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

A different language model for each document

language model of d_1

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.20	toad	.01
the	.20	said	.03
a	.10	likes	.02
frog	.01	that	.04
		...	...

language model of d_2

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.20	toad	.02
the	.15	said	.03
a	.08	likes	.02
frog	.01	that	.05
		...	...

query: frog said that toad likes frog STOP

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

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

$P(\text{query}|M_{d1}) < P(\text{query}|M_{d2})$: d_2 is more relevant to the query than d_1

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. by PageRank)
  - $P(q|d)$ is the probability of q given d.
- Under the assumptions we made, ranking documents according according to $P(q|d)$ and $P(d|q)$ is equivalent.

Where we are

- In the LM approach to IR, we 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 $P(q|d)$

- The conditional independence assumption:

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

  - $|q|$: length of q
  - t_k: the token occurring at position k in q

- This is equivalent to:

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

  - $\text{tf}_{t,q}$: term frequency (# occurrences) of t in q

Parameter estimation

- Missing piece: Where do the parameters $P(t|M_d)$ come from?
- Start with maximum likelihood estimates

$$\hat{P}(t|M_d) = \frac{\text{tf}_{t,d}}{|d|}$$

  - $|d|$: length of d
  - $\text{tf}_{t,d}$: # occurrences of t in d

- The zero problem (in nominator and denominator)
- A single t with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- Example: for query [Michael Jackson top hits] a document about "top songs" (but not with the word "hits") would have $P(q|M_d) = 0$
- We need to smooth the estimates to avoid zeros.

Smoothing

- Idea: A nonoccurring term is possible (even though it didn't occur)

  ...but no more likely than expected by chance in the collection.

- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

$$\hat{P}(t|M_c) = \frac{\text{cf}_t}{T}$$

  - M_c: the collection model
  - cf_t: the number of occurrences of t in the collection
  - $T = \sum_t \text{cf}_t$: the total number of tokens in the collection.

Jelinek-Mercer smoothing

- Intuition: Mixing the probability from the document with the general collection frequency of the word.

$$P(t|d) = \lambda P(t|M_d) + (1 - \lambda) P(t|M_c)$$

- High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words.
- Low value of λ: more disjunctive, suitable for long queries.
- Correctly setting λ 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:
  - 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: $d_2 > d_1$

Dirichlet smoothing

- Intuition: Before having seen any part of the document we start with the background distribution as our estimate.

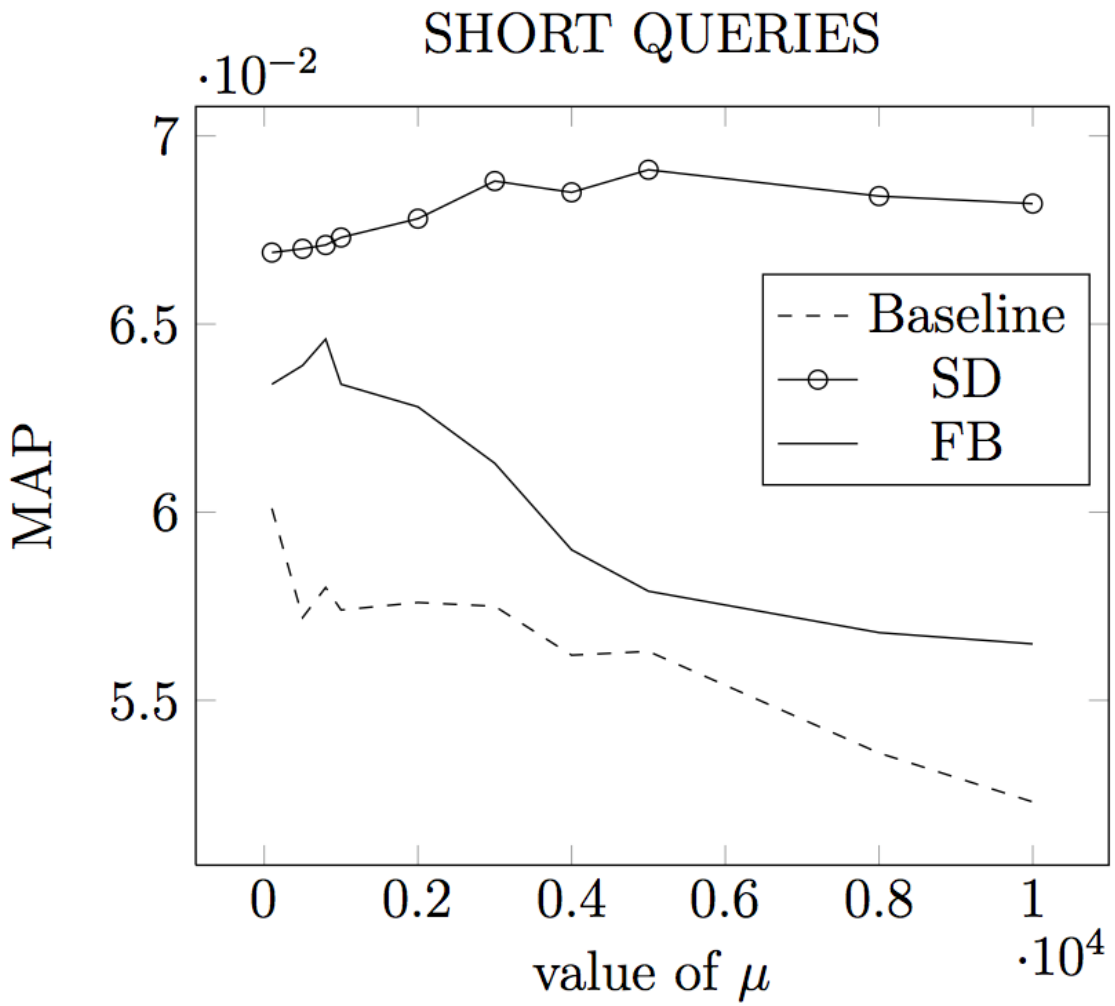
$$\hat{P}(t|d) = \frac{\text{tf}_{t,d} + \mu \hat{P}(t|M_c)}{L_d + \mu}$$

- The background distribution $\hat{P}(t|M_c)$ is the prior for $\hat{P}(t|d)$.
- As we read the document and count terms we update the background distribution.
- The weight factor μ 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



Language model vs. Vector space model: Example

	Precision			
Recall	TF-IDF	LM	%Δ	significant
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	
average	0.1868	0.2233	+19.6	*

The language modeling approach always does better in these experiments

...but significant gains are shown at higher levels of recall.

Language model vs. Vector space model: Things in common

1. Term frequency is directly in the model.
   - But it is not scaled in LMs.
2. Probabilities are inherently "length-normalized".
   - Cosine normalization does something similar for vector space.
3. Mixing document/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.

Language model vs. Vector space model: Differences

1. Language model: based on probability theory
2. Vector space: based on similarity, a geometric/linear algebra notion
3. Collection frequency vs. document frequency
4. Details of term frequency, length normalization etc.

Language models for IR: Assumptions

1. Queries and documents are objects of the same type.
   - There are other LMs for IR that do not make this assumption.
   - The vector space model makes the same assumption.
2. Terms are conditionally independent.
   - Vector space model (and Naive Bayes) make the same assumption.

- Language models have cleaner statement of assumptions and better theoretical foundation than vector space

... but "pure" LMs perform much worse than "tuned" LMs.

Text classification

A text classification task: Email spam filtering

```
From: ``'' <takworlld@hotmail.com>
Subject: real estate is the only way... gem  oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the
methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

=================================================
Click Below to order:
http://www.wholesaledaily.com/sales/nmd.htm
=================================================
```

How would you write a program that would automatically detect and delete this type of message?

Formal definition of TC: Training

Given:

- A document space $\mathbb{X}$
  - Documents are represented in this space – typically some type of high-dimensional space.
- A fixed set of classes $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$
  - The classes are human-defined for the needs of an application (e.g., spam vs. nonspam).
- A training set $\mathbb{D}$ of labeled documents. Each labeled document $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$

Using a learning method or learning algorithm, we then wish to learn a classifier γ that maps documents to classes:

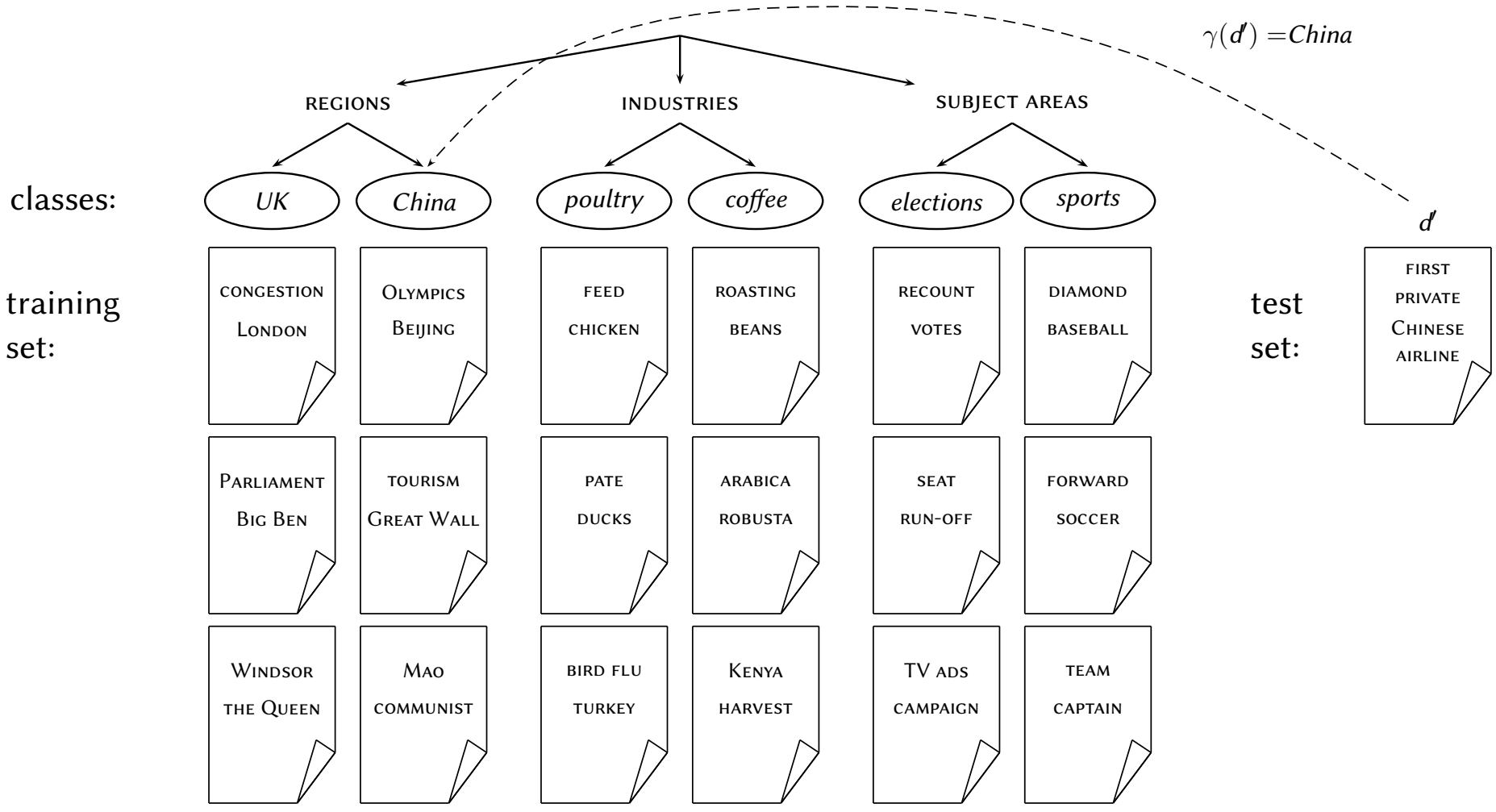
$$\gamma : \mathbb{X} \to \mathbb{C}$$

Formal definition of TC: Application/Testing

Given: a description $d \in \mathbb{X}$ of a document

Determine: $\gamma(d) \in \mathbb{C}$, that is, the class that is most appropriate for d

Topic classification



Examples of how search engines use classification

- Language identification (English vs. French, etc.)
- Detection of spam pages (spam vs. nonspam)
- Detection of sexually explicit content (sexually explicit vs. not)
- Topic-specific or *vertical* search (relevant to vertical vs. not)
- Sentiment detection (positive vs. negative)
- Machine-learned ranking function in ad hoc (relevant vs. nonrelevant)

Classification methods: 1. Manual

- Manual classification used by Yahoo in the beginning of the web
- Domain-specific classification, e.g. PubMed/MeSH
- Very accurate if job is done by experts
- Consistent when the problem size and team is small
- Scaling manual classification is difficult and expensive.

$\rightarrow$ We need automatic methods for classification.

Classification methods: 2. Rule-based

- E.g., Google Alerts is rule-based classification.
- There are IDE-type development enviroments for writing very complex rules efficiently. (e.g., Verity)
- Often: Boolean combinations (as in Google Alerts)
- Accuracy is very high if a rule has been carefully refined over time by a subject expert.
- Building and maintaining rule-based classification systems is cumbersome and expensive.

Classification methods: 3. Statistical/Probabilistic

- This was our original definition of the classification problem – text classification as a learning problem
- Tasks:
  i. Supervised learning of a the classification function γ
  ii. application of γ to classifying new documents
- Examples of methods for doing this: Naive Bayes and SVMs
- No free lunch: requires hand-classified training data
- But this manual classification can be done by non-experts.

Naive Bayes

The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

  - n_d – length of the document (number of tokens)
  - $P(t_k|c)$ – the probability of t_k occurring in a document of class c
  - $P(c)$ – the prior probability of c
- $P(t_k|c)$ measures how much evidence the term t_k contributes that c is the correct class of the document d.
- If a document's terms do not provide clear evidence for one class vs. another, we choose the c with highest $P(c)$.

Maximum a posteriori class

- Our goal in Naive Bayes classification is to find the "best" class.
- The best class is the most likely or maximum a posteriori class (MAP):

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

- We write $\hat{P}$ for P since these values are estimates from the training set.

Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, we can sum log probabilities instead of multiplying probabilities.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \left[\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c) \right]$$

Naive Bayes classifier

- Classification rule:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

- Simple interpretation:
  - Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator t_k is for c.
  - The prior $\log \hat{P}(c)$ is a weight indicating the relative frequency of c.
  - The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
  - We select the class with the most evidence.

Parameter estimation take 1: Maximum likelihood

- Estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from train data: How?
- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

  - N_c: number of docs in class c
  - N: total number of docs

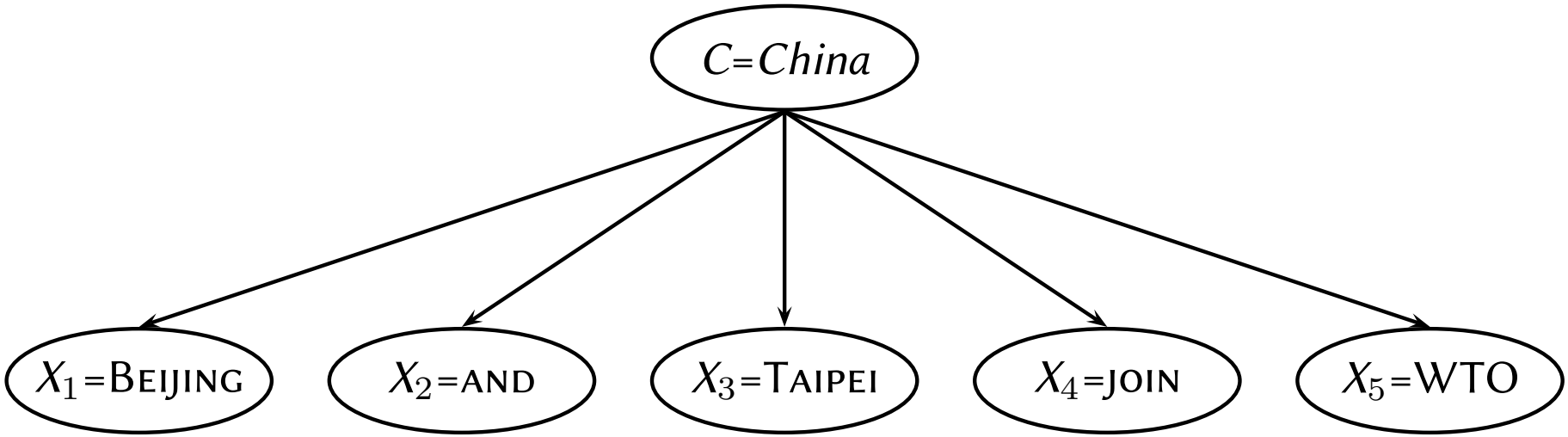
- Conditional:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

  - T_{ct} is the number of tokens of t in training documents from class c

- We've made a Naive Bayes independence assumption here: $\hat{P}(t_k|c) = \hat{P}(t_k|c)$, independent of position.

The problem with maximum likelihood estimates: Zeros



$$P(China|d) \propto P(China) \cdot P(\text{BEIJING}|China) \cdot P(\text{AND}|China) \cdot P(\text{TAIPEI}|China) \cdot P(\text{JOIN}|China) \cdot P(\text{WTO}|China)$$

- If WTO never occurs in class China in the train set:

$$\hat{P}(\text{WTO}|China) = \frac{T_{China,\text{WTO}}}{\sum_{t' \in V} T_{China,t'}} = \frac{0}{\sum_{t' \in V} T_{China,t'}} = 0$$

The problem with maximum likelihood estimates: Zeros (cont)

- If there are no occurrences of WTO in documents in class China, we get a zero estimate:

$$\hat{P}(\text{WTO}|China) = \frac{T_{China,\text{WTO}}}{\sum_{t' \in V} T_{China,t'}} = 0$$

→ We will get $P(China|d) = 0$ for any document that contains WTO!

To avoid zeros: Add-one smoothing

- Before:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- Now, add one to each count to avoid zeros:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

- B is (in this case) the number of different words or the size of the vocabulary $|V| = M$.

- For BIM we used "add 0.5" or ELE – we could also use that here.

Naive Bayes: Summary

- Estimate parameters from the training corpus by add-one smoothing
- For a new document, for each class, compute sum of
  (i) log of prior and
  (ii) logs of conditional probabilities of the terms
- Assign the document to the class with the largest score.

Naive Bayes: Training

TrainMultinomialNB($\mathbb{C}, \mathbb{D}$)

1 $V \leftarrow$ ExtractVocabulary($\mathbb{D}$)
2 $N \leftarrow$ CountDocs($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ CountDocsInClass($\mathbb{D}, c$)
5 $prior[c] \leftarrow N_c/N$
6 $text_c \leftarrow$ ConcatenateTextOfAllDocsInClass($\mathbb{D}, c$)
7 **for each** $t \in V$
8 **do** $T_{ct} \leftarrow$ CountTokensOfTerm($text_c, t$)
9 **for each** $t \in V$
10 **do** $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11 **return** $V, prior, condprob$

Naive Bayes: Testing

ApplyMultinomialNB($\mathbb{C}$, V, $prior$, $condprob$, d)

1 $W \leftarrow$ ExtractTokensFromDoc(V, d)
2 **for each** $c \in \mathbb{C}$
3 **do** $score[c] \leftarrow \log prior[c]$
4 **for each** $t \in W$
5 **do** $score[c] += \log condprob[t][c]$
6 **return** $\arg\max_{c \in \mathbb{C}} score[c]$

Time complexity of Naive Bayes

mode	time complexity
training	$\Theta(\|\mathbb{D}\|L_{\text{ave}} + \|\mathbb{C}\||V|)$
testing	$\Theta(L_a + \|\mathbb{C}\|M_a) = \Theta(\|\mathbb{C}\|M_a)$

- L_{ave}: average length of a training doc, L_a: length of the test doc, M_a: number of distinct terms in the test doc, $\mathbb{D}$: training set, V: vocabulary, $\mathbb{C}$: set of classes
- Training time is linear:
  - $\Theta(|\mathbb{D}|L_{\text{ave}})$ – time it takes to compute all counts.
  - $\Theta(|\mathbb{C}||V|)$ – time to compute the parameters from the counts.
  - Generally: $|\mathbb{C}||V| < |\mathbb{D}|L_{\text{ave}}$
- Test time is also linear (in the length of the test document).
- Thus: Naive Bayes is linear in the size of the training set (training) and the test document (testing). This is optimal.

Derivation of Naive Bayes rule

- We want to find the class that is most likely given the document:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} P(c|d)$$

- Apply Bayes rule $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \frac{P(d|c)P(c)}{P(d)}$$

- Drop denominator since $P(d)$ is the same for all classes:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} P(d|c)P(c)$$

Too many parameters / sparseness

$$
\begin{aligned}
c_{\text{map}} &= \arg\max_{c \in \mathbb{C}} P(d|c)P(c) \\
&= \arg\max_{c \in \mathbb{C}} P(\langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle | c) P(c)
\end{aligned}
$$

- There are too many parameters $P(\langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle | c)$, one for each unique combination of a class and a sequence of words.
- We would need a very, very large number of training examples to estimate that many parameters.
- This is the problem of data sparseness.

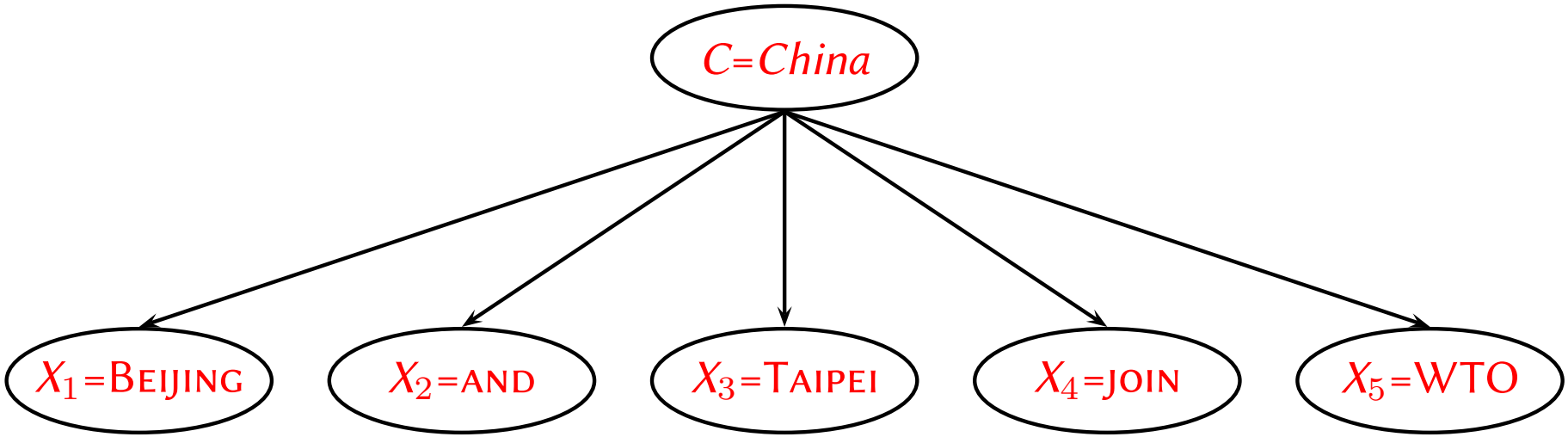
Naive Bayes conditional independence assumption

- To reduce the number of parameters to a manageable size, we make the Naive Bayes conditional independence assumption:

$$P(d|c) = P(\langle t_1, \ldots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

- We assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(X_k = t_k|c)$.
- Recall from earlier the estimates for these conditional probabilities: $\hat{P}(t|c) = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'})+B}$
- Difference to BIM? Will be discussed later.

Generative model



$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- Generate a class with probability $P(c)$
- Generate each of the words (in their respective positions), conditional on the class, but independent of each other, with probability $P(t_k|c)$
- To classify docs, we "reengineer" this process and find the class that is most likely to have generated the doc.

Second independence assumption

$$\hat{P}(X_{k_1} = t|c) = \hat{P}(X_{k_2} = t|c)$$

- For example, for a document in the class *UK*, the probability of generating QUEEN in the first position of the document is the same as generating it in the last position.
- The two independence assumptions amount to the bag of words model.

Violation of Naive Bayes independence assumptions

- Conditional independence:

$$P(\langle t_1, \ldots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

- Positional independence:

$$\hat{P}(X_{k_1} = t | c) = \hat{P}(X_{k_2} = t | c)$$

- The independence assumptions do not really hold of documents written in natural language.
- How can Naive Bayes work if it makes such inappropriate assumptions?

Why does Naive Bayes work?

- Naive Bayes can work well even though conditional independence assumptions are badly violated.
- Example:

	c_1	c_2	class selected
true probability $P(c\|d)$	0.6	0.4	c_1
$\hat{P}(c)\prod_{1\leq k\leq n_d}\hat{P}(t_k\|c)$	0.00099	0.00001	
NB estimate $\hat{P}(c\|d)$	0.99	0.01	c_1

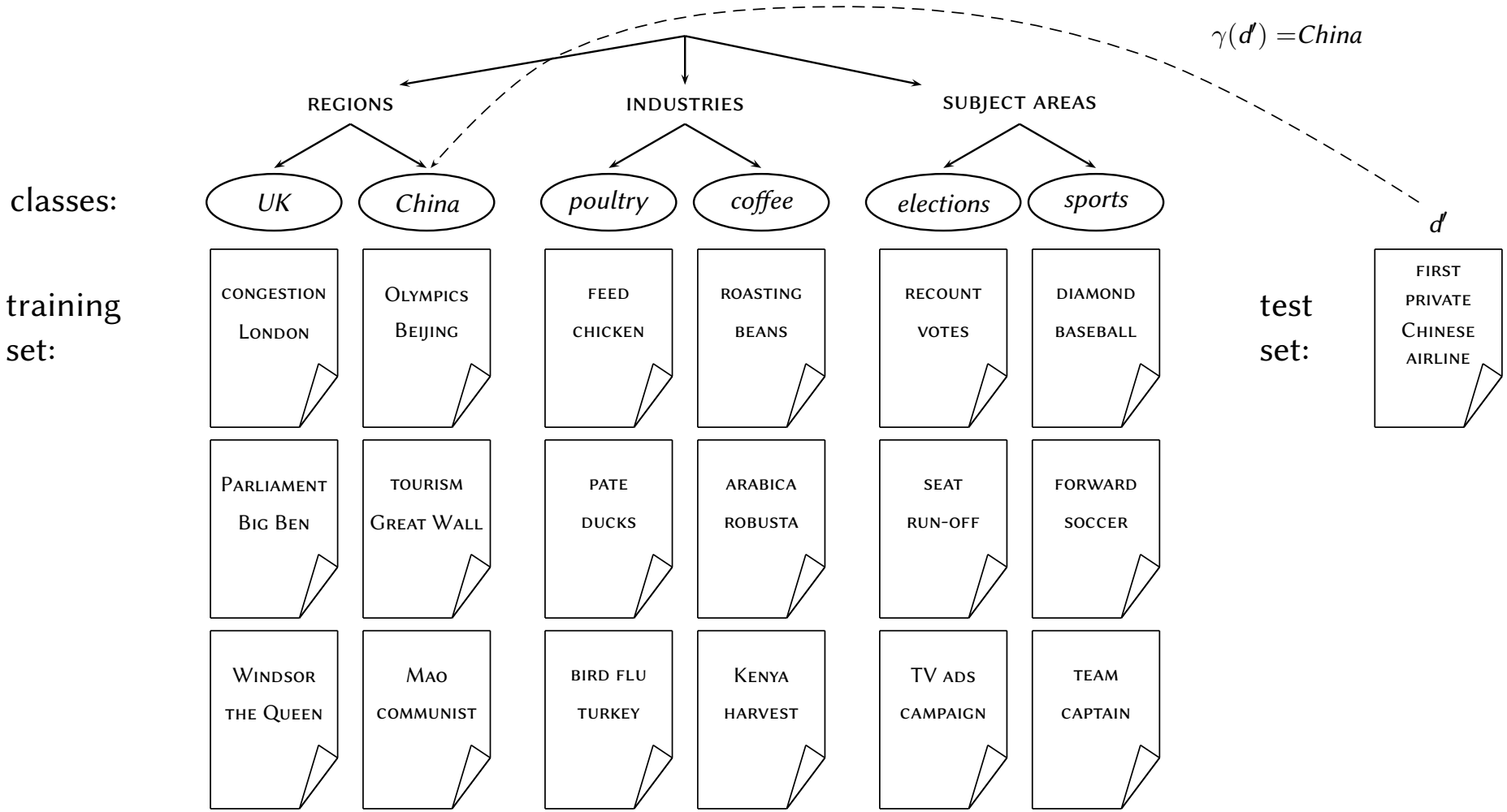
- Double counting of evidence causes underestimation (0.01) and overestimation (0.99).
- Classification is about predicting the correct class and not about accurately estimating probabilities.
- Naive Bayes is terrible for correct estimation but if often performs well at accurate prediction (choosing the correct class).

Naive Bayes is not so naive

- More robust to nonrelevant features than some more complex learning methods
- More robust to concept drift (changing of definition of class over time) than some more complex learning methods
- Better than methods like decision trees when we have many equally important features
- A good dependable baseline for text classification (but not the best)
- Optimal if independence assumptions hold (never true for text, but true for some domains)
- Very fast
- Low storage requirements

Evaluation of text classification

Evaluation on Reuters



Example: The Reuters collection

symbol	statistic	value
N	documents	800,000
L	avg. # word tokens per document	200
M	word types	400,000

type of class	number	examples
region	366	UK, China
industry	870	poultry, coffee
subject area	126	elections, sports

Evaluating classification

- Evaluation must be done on test data that are independent of the training data, i.e., training and test sets are disjoint.
- It's easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set).
- Measures: Precision, recall, F_1, classification accuracy

Precision P, recall R, and F_1 measure

	in the class	not in the class
predicted to be in the class	true positives (TP)	false positives (FP)
predicted to not be in the class	false negatives (FN)	true negatives (TN)

- TP, FP, FN, TN are counts of documents
- The sum of these four counts is the total number of documents.

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$
$$F_1 = \frac{1}{\frac{1}{2}\frac{1}{P} + \frac{1}{2}\frac{1}{R}} = \frac{2PR}{P + R}$$

- F_1 allows us to trade off precision against recall.

Averaging: Micro vs. Macro

- We now have an evaluation measure (F_1) for one class.
- But we also want a single number that measures the aggregate performance over all classes in the collection.
- Macroaveraging
  - Compute F_1 for each of the C classes
  - Average these C numbers
- Microaveraging
  - Compute TP, FP, FN for each of the C classes
  - Sum these C numbers (e.g., all TP to get aggregate TP)
  - Compute F_1 for aggregate TP, FP, FN

Naive Bayes vs. other methods (F_1)

(a)

	NB	Rocchio	kNN	SVM
micro-avg-L (90 classes)	80	85	86	89
macro-avg (90 classes)	47	59	60	60

(b)

	NB	Rocchio	kNN	trees	SVM
earn	96	93	97	98	98
acq	88	65	92	90	94
money-fx	57	47	78	66	75
grain	79	68	82	85	95
crude	80	70	86	85	89
trade	64	65	77	73	76
interest	65	63	74	67	78
ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
micro-avg (top 10)	82	65	82	88	92
micro-avg-D (118 classes)	75	62	n/a	n/a	87