Lecture 5: Language Modelling in Information Retrieval and Classification

Information Retrieval
Computer Science Tripos Part II

Helen Yannakoudakis[1]

Natural Language and Information Processing (NLIP) Group

UNIVERSITY OF CAMBRIDGE

helen.yannakoudakis@cl.cam.ac.uk

2018

[1]Based on slides from Simone Teufel and Ronan Cummins

Recap: Ranked retrieval in the vector space model

- Represent the query as a weighted tf–idf vector.
- Represent each document as a weighted tf–idf vector.
- Compute the cosine similarity between the query vector and each document vector.
- Rank documents with respect to the query.
- Return the top K (e.g., $K = 10$) to the user.

Upcoming today

- Query-likelihood method in IR
- Document Language Modelling
- Smoothing
- Classification

Overview

1. Query Likelihood
2. Estimating Document Models
3. Smoothing
4. Naive Bayes Classification

Language Model

- A model for how humans generate language.
- Places a probability distribution over any sequence of words.
- By construction, it also provides a model for generating text according to its distribution.
- Used in many language-orientated tasks, e.g.,
  - Machine translation:
    P(high winds tonite) > P(large winds tonite)
  - Spelling correction:
    P(about 15 minutes) > P(about 15 minuets)
  - Speech recognition:
    P(I saw a van) >> P(eyes awe of an)

How do we build probabilities over sequences of terms?

$$P(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2)P(t_4|t_1 t_2 t_3)$$

Unigram Language Model

How do we build probabilities over sequences of terms?

$$P(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2)P(t_4|t_1 t_2 t_3)$$

A unigram language model throws away all conditioning context, and estimates each term independently. As a result:

$$P_{uni}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2)P(t_3)P(t_4)$$

What is a document language model?

- A model for how an author generates a document on a particular topic.
- The document itself is just one sample from the model (i.e., ask the author to write the document again and he/she will invariably write something similar, but not exactly the same).
- A probabilistic generative model for documents.

Two Unigram Document Language Models

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$\sum_{t \in V} P(t|M_d) = 1$$

Query Likelihood Method (I)

- Users often pose queries by thinking of words that are likely to be in *relevant* documents.
- The query likelihood approach uses this idea as a principle for ranking documents.
- We construct from each document d in the collection a language model M_d.
- Given a query string q, we rank documents by the likelihood of their document *models* M_d generating q: $P(q|M_d)$

Query Likelihood Method (II)

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

Query Likelihood Method (II)

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

$$P(d|q) \propto P(q|d)P(d)$$

Query Likelihood Method (II)

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

$$P(d|q) \propto P(q|d)P(d)$$

where if we have a uniform prior over $P(d)$ then

$$P(d|q) \propto P(q|d)$$

Note: $P(d)$ is uniform if we have no reason a priori to favour one document over another. Useful priors (based on aspects such as authority, length, novelty, freshness, popularity, click-through rate) could easily be incorporated.

An Example (I)

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$P(\mathit{frog}\ \mathit{said}\ \mathit{that}\ \mathit{toad}\ \mathit{likes}\ \mathit{frog}|M_1) =$$

An Example (I)

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$P(frog\ said\ that\ toad\ likes\ frog|M_1) = (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01)$$

An Example (I)

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$P(frog\ said\ that\ toad\ likes\ frog|M_1) = (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01)$$

$$P(frog\ said\ that\ toad\ likes\ frog|M_2) =$$

An Example (I)

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$P(frog\ said\ that\ toad\ likes\ frog|M_1) = (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01)$$

$$P(frog\ said\ that\ toad\ likes\ frog|M_2) = (0.0002 \times 0.03 \times 0.04 \times 0.0001 \times 0.04 \times 0.0002)$$

An Example (II)

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

$$P(q|M_1) > P(q|M_2)$$

Overview

Documents as samples

- We now know how to rank document models in a theoretically principled manner.
- But how do we estimate the document model for each document?

Documents as samples

- We now know how to rank document models in a theoretically principled manner.
- But how do we estimate the document model for each document?

Example document

click go the shears boys click click click

Documents as samples

- We now know how to rank document models in a theoretically principled manner.
- But how do we estimate the document model for each document?

Example document

click go the shears boys click click click

Maximum likelihood estimate (MLE)

Estimating the probability as the relative frequency of t in d: $\frac{tf_{t,d}}{|d|}$ for the unigram model ($|d|$: length of the document)

Documents as samples

- We now know how to rank document models in a theoretically principled manner.
- But how do we estimate the document model for each document?

Example document

click go the shears boys click click click

Maximum likelihood estimate (MLE)

Estimating the probability as the relative frequency of t in d: $\frac{tf_{t,d}}{|d|}$ for the unigram model ($|d|$: length of the document)

Maximum likelihood estimates

click=$\frac{4}{8}$, go=$\frac{1}{8}$, the=$\frac{1}{8}$, shears=$\frac{1}{8}$, boys=$\frac{1}{8}$

Zero probability problem (over-fitting)

- But when using maximum likelihood estimates, documents that do not contain *all* query terms will receive a score of zero.

Zero probability problem (over-fitting)

- But when using maximum likelihood estimates, documents that do not contain *all* query terms will receive a score of zero.

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Zero probability problem (over-fitting)

- But when using maximum likelihood estimates, documents that do not contain *all* query terms will receive a score of zero.

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Sample query

$P(shears\ boys\ hair|M_d) = 0.125 \times 0.125 \times 0 = 0$ (*hair* is an unseen word)

What if the query is long?

Problem in calculation of estimation

- With MLE, only seen terms receive a probability estimate.
- The total probability attributed to the seen terms is 1.

Problem in calculation of estimation

- With MLE, only seen terms receive a probability estimate.
- The total probability attributed to the seen terms is 1.
- Remember that the document model is a generative explanation.
- The document itself is just one sample from the model. If a person was to rewrite the document, he/she may include *hair* or indeed some other words.

Problem in calculation of estimation

- With MLE, only seen terms receive a probability estimate.
- The total probability attributed to the seen terms is 1.
- Remember that the document model is a generative explanation.
- The document itself is just one sample from the model. If a person was to rewrite the document, he/she may include *hair* or indeed some other words.
- The estimated probabilities of seen terms is too big!
- MLE overestimates the probability of seen terms.

Problem in calculation of estimation

- With MLE, only seen terms receive a probability estimate.
- The total probability attributed to the seen terms is 1.
- Remember that the document model is a generative explanation.
- The document itself is just one sample from the model. If a person was to rewrite the document, he/she may include *hair* or indeed some other words.
- The estimated probabilities of seen terms is too big!
- MLE overestimates the probability of seen terms.

Solution: smoothing

Take some portion away from the MLE overestimate, and re-distribute it to the unseen terms.

Solution: smoothing

Discount non-zero probabilities and to give some probability mass to unseen words:

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Solution: smoothing

Discount non-zero probabilities and to give some probability mass to unseen words:

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Some type of smoothing

click=0.4, go=0.1, the=0.1, shears=0.1, boys=0.1, hair=0.01, man=0.01, the=0.001, bacon=0.0001, ...

Overview

How to smooth

ML estimates:

$$\hat{P}(t|M_d) = \frac{tf_{t,d}}{|d|}$$

How to smooth

ML estimates:

$$\hat{P}(t|M_d) = \frac{tf_{t,d}}{|d|}$$

Linear Smoothing:

$$\hat{P}(t|M_d) = \lambda \frac{tf_{t,d}}{|d|} + (1-\lambda)\hat{P}(t|M_c)$$

M_c is a language model built from the entire document collection.

$\hat{P}(t|M_c) = \frac{cf_t}{|c|}$ is the estimated probability of seeing t in general (i.e., cf_t is the frequency of t in the entire document collection of $|c|$ tokens).

λ is a smoothing parameter between 0 and 1.

How to smooth

Linear Smoothing:

$$\hat{P}(t|M_d) = \lambda \frac{tf_{t,d}}{|d|} + (1-\lambda)\frac{cf_t}{|c|}$$

- High λ: more conjunctive search (i.e., where we retrieve documents containing all query terms).
- Low λ: more disjunctive search (suitable for long queries).
- Correctly setting λ is important to the good performance of the model (collection-specific tuning).
- Note: every document has the same amount of smoothing.

How to smooth

Linear Smoothing:

$$\hat{P}(t|M_d) = \lambda \frac{tf_{t,d}}{|d|} + (1-\lambda)\frac{cf_t}{|c|}$$

How to smooth

Linear Smoothing:

$$\hat{P}(t|M_d) = \lambda \frac{tf_{t,d}}{|d|} + (1-\lambda)\frac{cf_t}{|c|}$$

Dirichlet Smoothing has been found to be more effective in IR where $\lambda = \frac{|d|}{\alpha + |d|}$

- Dynamic smoothing that changes based on the document length.

How to smooth

Linear Smoothing:

$$\hat{P}(t|M_d) = \lambda \frac{tf_{t,d}}{|d|} + (1-\lambda)\frac{cf_t}{|c|}$$

Dirichlet Smoothing has been found to be more effective in IR where $\lambda = \frac{|d|}{\alpha+|d|}$

- Dynamic smoothing that changes based on the document length.

Plugging this in yields:

$$\hat{P}(t|M_d) = \frac{|d|}{\alpha+|d|}\frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha+|d|}\frac{cf_t}{|c|}$$

where α can be interpreted as the background mass (total number of pseudo counts of words introduced).

Bayesian Intuition

We should have more trust (belief) in ML estimates that are derived from longer documents – see the $\frac{|d|}{\alpha+|d|}$ factor.

Putting this all together

Rank documents according to:

$$P(q|d) = \prod_{t \in q} \left(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|} \right)$$

or

Putting this all together

Rank documents according to:

$$P(q|d) = \prod_{t \in q} \left(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|} \right)$$

or

$$\log P(q|d) = \sum_{t \in q} log \left(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|} \right)$$

Putting this all together

Rank documents according to:

$$P(q|d) = \prod_{t \in q} (\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|})$$

or

$$\log\ P(q|d) = \sum_{t \in q} log(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|})$$

In practice, we use logs – why?

Putting this all together

Rank documents according to:

$$P(q|d) = \prod_{t \in q} \left(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|}\right)$$

or

$$\log\ P(q|d) = \sum_{t \in q} log\left(\frac{|d|}{\alpha + |d|} \frac{tf_{t,d}}{|d|} + \frac{\alpha}{\alpha + |d|} \frac{cf_t}{|c|}\right)$$

In practice, we use logs – why?

Multiplying lots of small probabilities can result in floating point underflow [$\log(xy) = \log(x) + \log(y)$].

Pros and Cons

- It is principled, intuitive, simple, and extendable.
- Aspects of *tf* and *idf* are incorporated quite naturally.
- It is computationally efficient for large scale corpora.
- More complex language models (markov models) can be adopted and document priors can be added.
- But more complex models usually involve storing more parameters (and doing more computation).

Pros and Cons

- It is principled, intuitive, simple, and extendable.
- Aspects of *tf* and *idf* are incorporated quite naturally.
- It is computationally efficient for large scale corpora.
- More complex language models (markov models) can be adopted and document priors can be added.
- But more complex models usually involve storing more parameters (and doing more computation).

- Both documents and queries are modelled as simple strings of symbols.
- No formal treatment of relevance.
- Therefore model does not handle *relevance feedback* automatically (lecture 7).

Extensions

- Relevance-based language models (very much related to Naive-Bayes classification) incorporate the idea of relevance and are useful for capturing feedback.
- Treating the query as being drawn from a query model (useful for long queries).
- Markov-chain models for document modelling.
- Use different generative distributions (e.g., replacing the multinomial with neural models).

Very useful resource:
`http://times.cs.uiuc.edu/czhai/pub/slmir-now.pdf`

Overview

Terminology

- Features: measurable properties of the data.
- Classes: labels associated with the data.
  - Sentiment classification: automatically classify text based on the sentiment it contains (e.g., movie reviews).
  - Features: the words the text contains, parts of speech, grammatical constructions etc.
  - Classes: positive or negative sentiment (binary classification).
- Classification is the function that maps input features to a class.

Examples of how search engines use classification

- Query classification (types of queries)
- Spelling correction
- Document/webpage classification
- Automatic detection of spam pages (spam vs. non-spam)
- Topic classification (relevant to topic vs. not)
- Language identification (classes: English vs. French etc.)
- User classification (personalised search)

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 follows:

$$P(c|d) = \frac{P(c)\,P(d|c)}{P(d)}$$

$$P(c|d) \propto P(c)\,P(d|c)$$

$P(d)$ is constant during a given classification and won't affect the result.

The Naive Bayes classifier

$$P(c|d) \propto P(c)\, P(d|c)$$

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

- $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c (conditional independence assumption).
- $|d|$ is the length of the document (number of tokens).
- $P(c)$ is the prior probability of c.
- 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 (MAP) class c_{map}:

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

Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

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

Naive Bayes classifier: Interpretation

- Classification rule:

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

- Simple interpretation:
  - Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator term t_k is for class c.
  - The prior $\log \hat{P}(c)$ is a weight that indicates how likely we are to see class 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 training data – how?

Parameter estimation take 1: Maximum likelihood

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

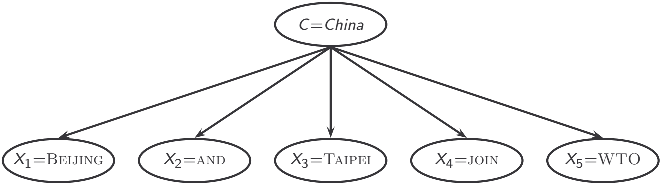
$$\hat{P}(c) = \frac{N_c}{N}$$

- N_c: number of docs in class c; N: total number of docs
- Conditional probabilities:

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

- T_{ct} is the number of times t occurs in training documents that belong to class c (includes multiple occurrences).
- We've made a Naive Bayes independence assumption here: $\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c)$, independent of positions k_1, k_2.

The problem with maximum likelihood estimates: Zeros

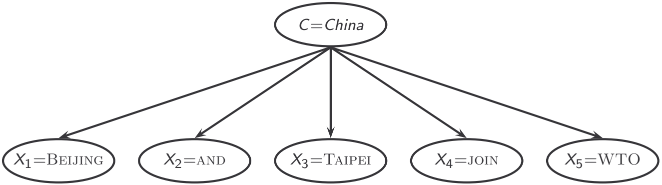


$$
\begin{aligned}
P(China|d) \quad \propto \quad & P(China) \cdot P(\textsc{Beijing}|China) \cdot P(\textsc{and}|China) \\
& \cdot P(\textsc{Taipei}|China) \cdot P(\textsc{join}|China) \cdot P(\mathrm{WTO}|China)
\end{aligned}
$$

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

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

The problem with maximum likelihood estimates: Zeros



$$
\begin{aligned}
P(China|d) \quad \propto \quad & P(China) \cdot P(\text{BEIJING}|China) \cdot P(\text{AND}|China) \\
& \cdot P(\text{TAIPEI}|China) \cdot P(\text{JOIN}|China) \cdot P(\text{WTO}|China)
\end{aligned}
$$

- If WTO never occurs in class China in the training 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

- If there are no occurrences of WTO in documents in class China ...
- ...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:

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

where V is the vocabulary of all distinct words, no matter which class c a term t occurred with.

Example

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

- Estimate parameters of Naive Bayes classifier using the training set.
- Classify test document.

$|text_c| = 8$ (no of tokens in class China)
$|text_{\overline{c}}| = 3$ (no of tokens in other class)
$|V| = 6$ (vocabulary size)

Example: Parameter estimates

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$

Conditional probabilities:

$$\begin{aligned}
\hat{P}(\textsc{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\textsc{Tokyo}|c) = \hat{P}(\textsc{Japan}|c) &= (0+1)/(8+6) = 1/14
\end{aligned}$$

$$\begin{aligned}
\hat{P}(\textsc{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\textsc{Tokyo}|\overline{c}) = \hat{P}(\textsc{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the vocabulary consists of 6 terms.

Example: Classification

$$\begin{aligned}\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003 \\ \hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001\end{aligned}$$

Thus, the classifier assigns the test document to $c = China$.
The reason for this classification decision is that the three occurrences of the positive indicator CHINESE in d_5 outweigh the occurrences of the two negative indicators JAPAN and TOKYO.

Naive Bayes is not so naive

- Multinomial model violates two independence assumptions and yet...
- Naive Bayes has won some competitions (e.g., KDD-CUP 97; prediction of most likely donors for a charity)
- More robust to non-relevant 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

Time complexity of Naive Bayes

mode	time complexity
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}} + \lvert\mathbb{C}\rvert\lvert V\rvert)$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{C}\rvert M_{\text{a}}) = \Theta(\lvert\mathbb{C}\rvert M_{\text{a}})$

- 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
- $\Theta(|\mathbb{D}|L_{\text{ave}})$ is the time it takes to compute all counts. Note that $|\mathbb{D}|L_{\text{ave}}$ is T, the size of our collection.
- $\Theta(|\mathbb{C}||V|)$ is the time it takes to compute the conditional probabilities 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.

Not covered

- Evaluation of text classification.

Summary

- Query-likelihood as a general principle for ranking documents in an unsupervised manner
  - Treat queries as strings
  - Rank documents according to their models
- Document language models
  - Know the difference between the document and the document model
  - Multinomial distribution is simple but effective
- Smoothing
  - Reasons for, and importance of, smoothing
  - Dirichlet (Bayesian) smoothing is very effective
- Classification
  - Text classification is supervised learning
  - Naive Bayes: simple baseline text classifier

Reading

- Manning, Raghavan, Schütze: Introduction to Information Retrieval (MRS), chapter 12: Language models for information retrieval
- MRS chapters 13.1-13.4 for text classification