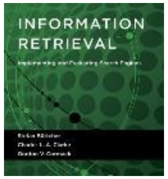
Vector Space Model and Language Models

VSM, LM Jelinek-Mercer Smoothing and LM Dirichlet Smoothing

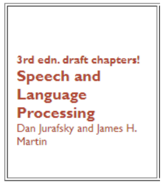
Information Retrieval

Retrieval models from:

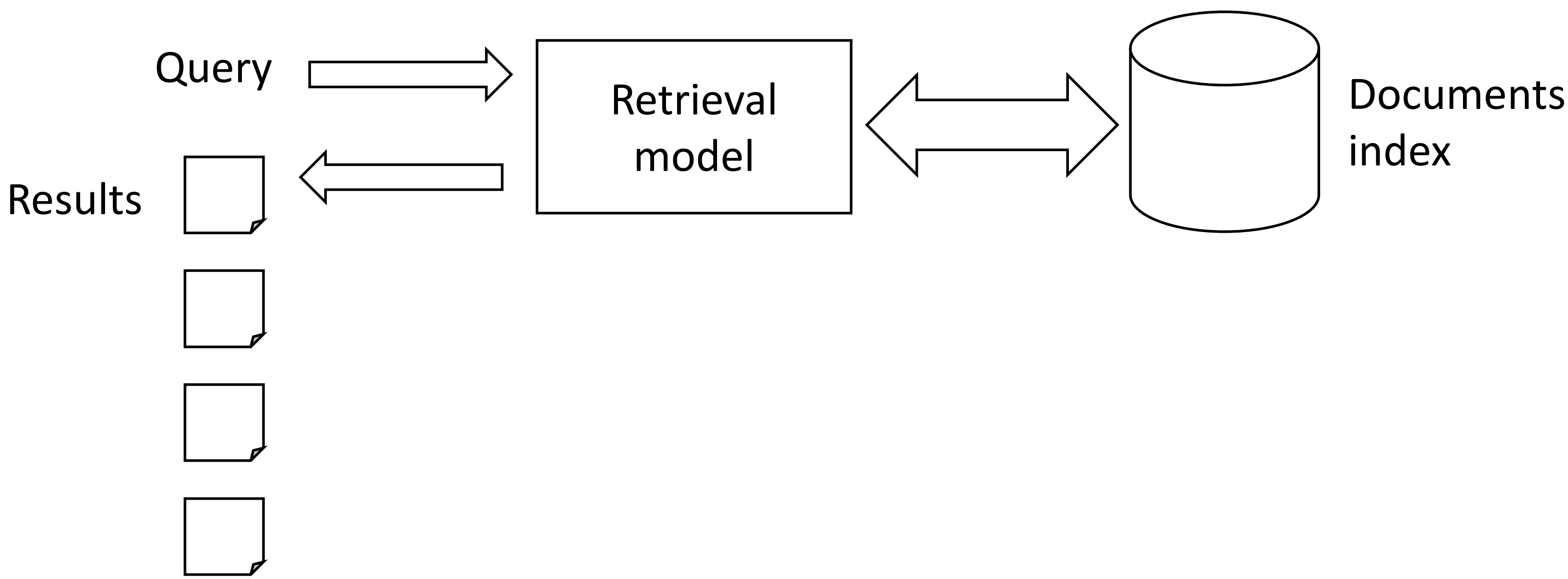
Experimental results from:



Bigram LM example from:



IR Problem setting



Retrieval models

- Geometric/linear spaces
  - Vector space model
- Language models approach to IR
  - Language models and smoothing
- Probabilistic retrieval model
  - Binary independence model
  - Okapi's BM25

Bag of Words representation

- After the text analysis steps, a document is represented as a vector of unigrams, n-grams, named entities, etc.

$$d = (\underbrace{w_1, \dots, w_L}_{\text{Unigrams}}, \underbrace{ng_1, \dots, ng_M}_{\text{Bigrams}}, \underbrace{ne_1, \dots, ne_N}_{\text{Named entities}})$$

- Each dimension indicates the presence of that particular unigram, n-gram, etc.
- The scale of that dimension indicates the importance of that unigram in the document.

Term weighting

- Boolean retrieval looks for terms overlap
- What’s wrong with the overlap measure?
- It doesn’t consider:
  - Term frequency in document
  - Term scarcity in collection (document mention frequency)
    - ***of*** is more common than ***ideas*** or ***march***
  - Length of documents
    - (And queries: score not normalized)

Term weighting

- Term weighting tries to reflect the importance of a document for a given query term.

- Term weighting must consider two aspects:
  - The frequency of a term in a document
  - Consider the rarity of a term in the repository

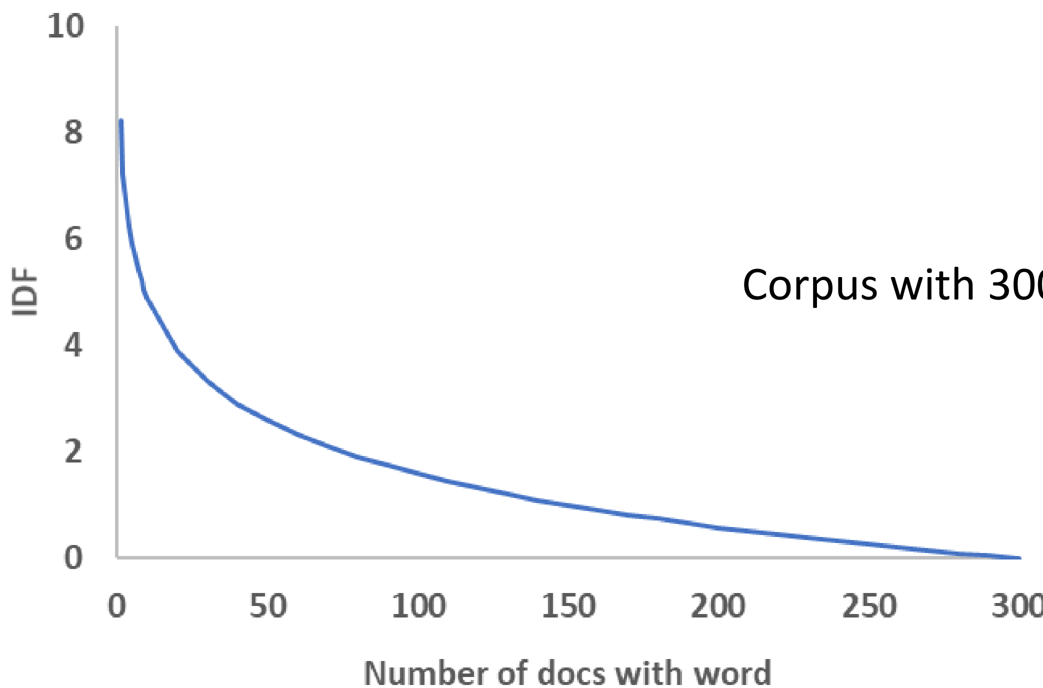
- Several weighting intuitions were evaluated throughout the years:
  - Salton, G. and Buckley, C. 1988. **Term-weighting approaches in automatic text retrieval**. *Inf. Process. Manage.* 24, 5 (Aug. 1988), 513-523.
  - Robertson, S. E. and Sparck Jones, K. 1988. **Relevance weighting of search terms**. In *Document Retrieval Systems*, P. Willett, Ed. Taylor Graham Series In Foundations Of Information Science, vol. 3.

Term Freq.-Inverted Document Freq.

- Text terms should be weighted according to
  - their importance for a given document: $tf_i(d) = n_i(d)$
  - and how rare a word is: $idf_i = log \frac{|D|}{df(t_i)}$ $df(t_i) = |\{d: t_i \in d\}|$
- The final **tf-idf** term weight is: $w_{i,j} = \mathrm{tf_i}(\mathrm{d_j}) \cdot idf_i$

Chapter 6: “**Introduction to Information Retrieval**”, Cambridge University Press, 2008

Inverse document frequency



Corpus with 300 documents

$$idf_i = log\frac{|D|}{df(t_i)} \qquad df(t_i) = |\{d: t_i \in d\}|$$

Chapter 6: "**Introduction to Information Retrieval**", Cambridge University Press, 2008

Vector Space Model

- Each doc d can now be viewed as a vector of tf $\times$ idf values, one component for each term

- So, we have a vector space where:
  - terms are axes
  - docs live in this space
  - even with stemming, it may have 50,000+ dimensions

- First application: Query-by-example
  - Given a doc d, find others “like” it.

- Now that d is a vector, find vectors (docs) “near” it.

Salton, G., Wong, A., and Yang, C. S. 1975. A vector space model for automatic indexing. Commun. ACM 18, 11 (Nov. 1975).

Documents and queries

- Documents are represented as an histogram of terms, n-grams and other indicators:

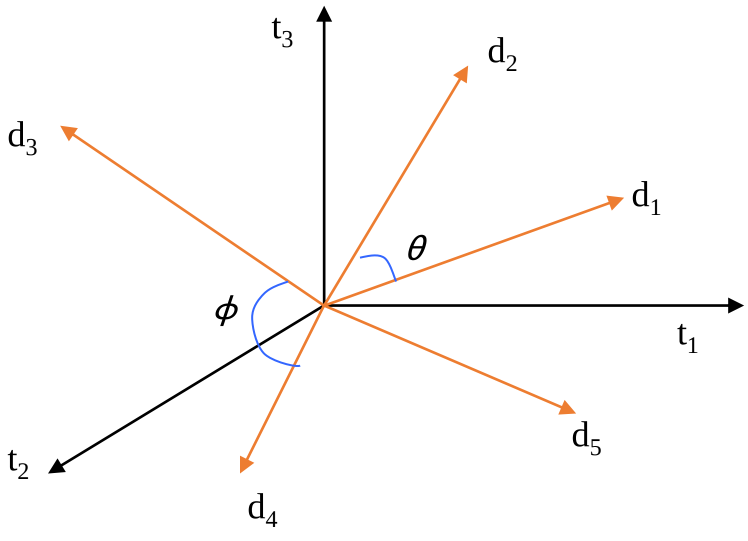
$$d = (w_1, \ldots, w_L, ng_1, \ldots, ng_M, PR, \ldots)$$

- The text query is processed with the same text pre-processing techniques.
  - A query is then represented as a vector of text terms and n-grams (and possibly other indicators):

$$\mathrm{q} = (w_1, \ldots, w_L, ng_1, \ldots, ng_M)$$

Intuition

- If d_1 is near d_2, then d_2 is near d_1.
- If d_1 near d_2, and d_2 near d_3, then d_1 is not far from d_3.
- No doc is closer to d than d itself.
    - Postulate: Documents that are "close together" in the vector space talk about the same things.



First cut

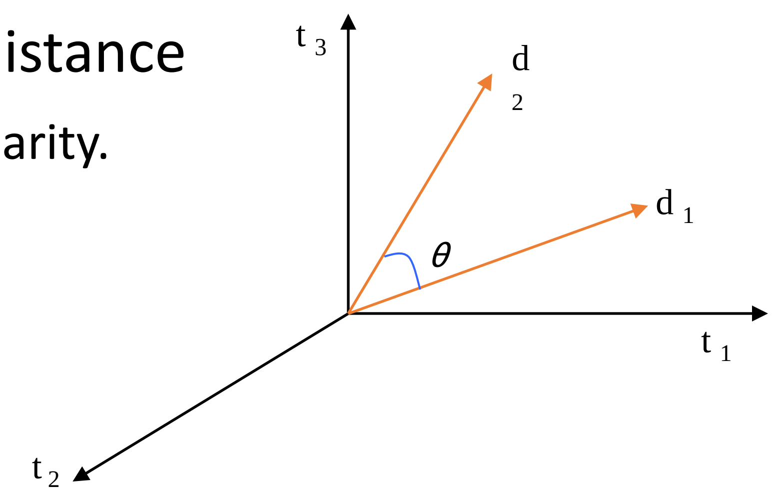
- Idea: Distance between d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance

- Why is this not a great idea?

Angle as a similarity

- Distance between vectors d_1 and d_2 captured by the cosine of the angle x between them.
  - Vectors pointing in the same direction

- Note – this is similarity, not distance
  - No triangle inequality for similarity.



Vectors normalization

- A vector can be normalized (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

- This maps vectors onto the unit sphere. Then, $|d_j| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$

- Longer documents don't get more weight

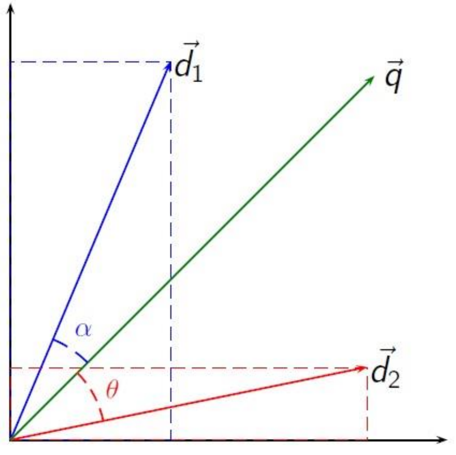
Chapter 6: "**Introduction to Information Retrieval**", Cambridge University Press, 2008

Cosine similarity

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

$$sim(q, d_i) = \cos(q, d_i) = \frac{q \cdot d_i}{\|q\|\|d_i\|}$$

$$sim(q, d_i) = \cos(q, d_i) = \frac{\sum_t q_t \cdot d_{i,t}}{\sqrt{\sum_t q_t^2}\sqrt{\sum_t d_{i,t}^2}}$$

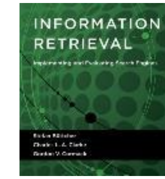


Improved semantics

- How to enhace/improve the previous model?
  - Positional indexing
    - Documents with distances between query terms greater than n are discarded
    - Distance between query terms in the documents affect rank score
  - Other ranking functions
    - BM-25
    - Bayesian networks
    - Learning to rank

Chapter 6: “**Introduction to Information Retrieval**”, Cambridge University Press, 2008

Experimental comparison



	TREC45				Gov2			
	1998		1999		2005		2006	
Method	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
Cosine TF-IDF	0.264	0.126	0.252	0.135	0.120	0.060	0.194	0.092
Proximity	0.396	0.124	0.370	0.146	0.425	0.173	0.562	0.23
No length norm. (rawTF)	0.266	0.106	0.240	0.120	0.298	0.093	0.282	0.097
D: rawTF+ noIDF Q: IDF	0.342	0.132	0.328	0.154	0.400	0.144	0.466	0.151

Probability Ranking Principle

Information Retrieval

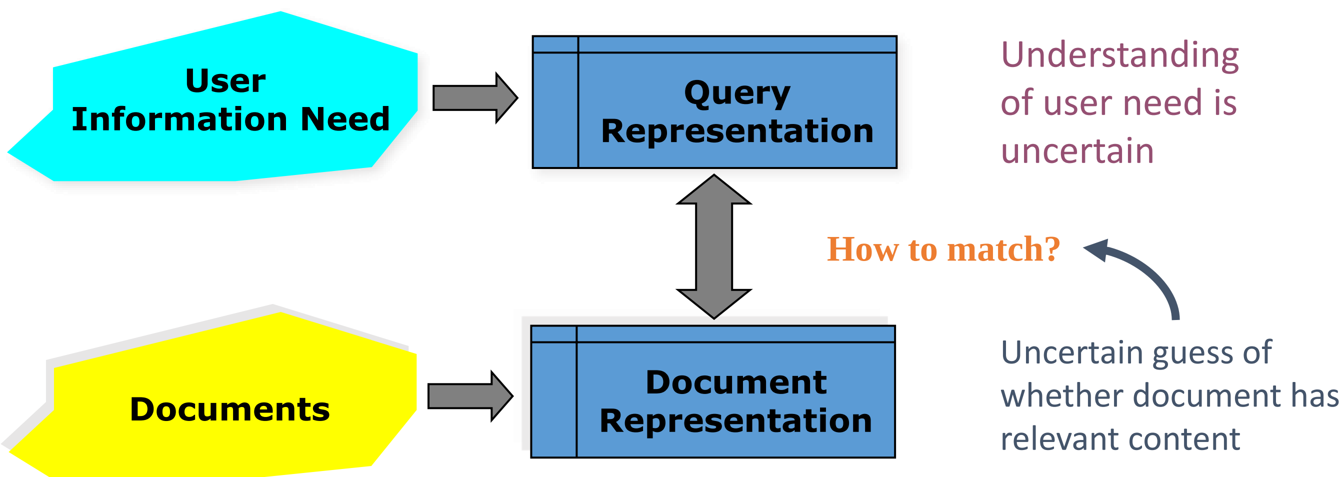
The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned
- Ranking method is the core of an IR system:
  - In what order do we present documents to the user?
  - We want the “best” document to be first, second, etc….

Idea: Rank by probability of relevance of the document w.r.t. information need

Why probabilities in IR?

- In traditional IR systems, matching between each document and query is attempted in a **semantically imprecise space** of index terms.



Probabilities provide a principled foundation for reasoning about uncertainty.

Modeling relevance

P(R=1|document, query)

- Let **d** represent a document in the collection.
- Let **R** represent relevance of a document w.r.t. to a query **q**
- Let **R=1** represent relevant and **R=0** not relevant.

- Our goal is to estimate: $$p(r=1|q,d) = \frac{p(d,q|r=1)p(r=1)}{p(d,q)}$$

$$p(r=0|q,d) = \frac{p(d,q|r=0)p(r=0)}{p(d,q)}$$

Probability Ranking Principle (PRP)

- PRP in action: Rank all documents by $p(r = 1|q, d)$
  - Theorem: Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss
  - Provable if all probabilities correct, etc. [e.g., Ripley 1996]

$$p(r|q, d) = \frac{p(d, q|r)p(r)}{p(d, q)}$$

- Using odds, we reach a more convenient formulation of ranking :

$$\boxed{\mathrm{O}(R|q, d) = \frac{p(r = 1|q, d)}{p(r = 0|q, d)}}$$

Language models interpretation

- PRP in action: Rank all documents by $p(r = 1|q, d)$
  - Theorem: Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss
  - Provable if all probabilities correct, etc. [e.g., Ripley 1996]

$$p(r|q, d) = \frac{p(d, q|r)p(r)}{p(d, q)}$$

- Using odds, we reach a more convenient ranking formulation:

$$\mathrm{O}(R|q, d) = \frac{p(r = 1|q, d)}{p(r = 0|q, d)} = \frac{\frac{p(q, d|r = 1)p(r = 1)}{p(d, q)}}{\frac{p(q, d \mid r = 0)p(r = 0)}{p(d, q)}} \propto \log \frac{p(q|d, r\ )p(r|d)}{p(q|d, \bar{r})p(\bar{r}|d)}$$

Language models

- In language models, we do a formulation towards the query posterior given the document as a model.

$$\mathrm{O}(R|q,d) = \frac{p(r=1|q,d)}{p(r=0|q,d)} = \frac{\frac{p(d,q|r=1)p(r=1)}{p(d,q)}}{\frac{p(d,q|r=0)p(r=0)}{p(d,q)}}$$

$$\propto \log \frac{p(q|d,r\ )p(r|d)}{p(q|d,\bar{r})p(\bar{r}|d)} \ =$$

$$= \log p(q|d,r) - \log p(q|d,\bar{r}) + \log \frac{p(r|d)}{p(\bar{r}|d)}$$

Language models

$$\log p(q|d,r) - \log p(q|d,\bar{r}) + \log \frac{p(r|d)}{p(\bar{r}|d)}$$

$$\boxed{\approx \log p(q|d,r) + \operatorname{logit}\big(p(r|d)\big)}$$

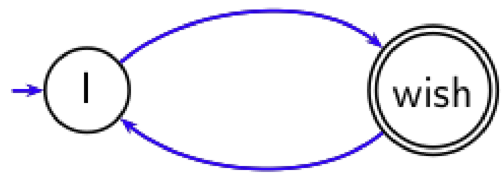
- The fist term computes the probability that the query has been generated by the document model
- The second term can measure the quality of the document with respect to other indicators not contained in the query (e.g. PageRank or number of links)

Language Models

Information Retrieval

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.

Types of language models

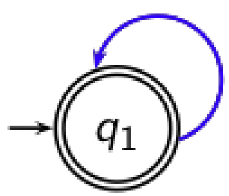
- Unigrams: $p_{uni}(t_1 t_2 t_3 t_4) = p(t_1)p(t_2)p(t_3)p(t_4)$

- Bigrams: $p_{bi}(t_1 t_2 t_3 t_4) = p(t_1)p(t_2|t_1)p(t_3|t_2)p(t_4|t_3)$

- Longer n-grams.

A probabilistic language model

- This is a one-state probabilistic finite-state automaton (a unigram LM) and the state emission distribution for its one state q1.
  - STOP is not a word, but a special symbol indicating that the automaton stops.



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
		...	...

String = “frog said that toad likes frog STOP”

P(string) = 0.01 · 0.03 · 0.04 · 0.01 · 0.02 · 0.01 · 0.2= 0.0000000000048

A language model per 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
		...	...

String = “frog said that toad likes frog STOP”

P(string|Md1) = $0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2 = 0.0000000000048 = 4.8 \cdot 10^{-12}$

P(string|Md2) = $0.01 \cdot 0.03 \cdot 0.05 \cdot 0.02 \cdot 0.02 \cdot 0.01 \cdot 0.2 = 0.0000000000120 = 12 \cdot 10^{-12}$

P(string|Md1) < P(string|Md2)

- Thus, document d2 is “more relevant” to the string “frog said that toad likes frog STOP” than d1 is.

Document models M_d

- Count the number of word occurrences
- Divide by the length of the 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
		...	...

Document bigram models M_d

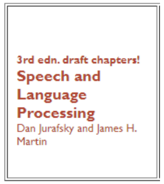


Figure 3.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

How to compute $p(q|d)$?

- We will make the same conditional independence assumption as for Naive Bayes (we dropped the *r* variable)

$$p(q|M_d) = \prod_{i=0}^{|q|} p(t_i|M_d)$$

  - |q| length of query;
  - t_i the token occurring at position *i* in the query

- This is equivalent to: $p(q|M_d) = \prod_{t \in \{q \cap d\}} p(t|M_d)^{tf_{t,q}}$

  - $tf_{t,q}$ is the term frequency (# occurrences) of t in q

- Multinomial model (omitting constant factor)

Parameter estimation

- The parameters $p(t|M_d)$ are obtained from the document data as the maximum likelihood estimate:

$$p\left(t\middle|M_d^{ml}\right) = \frac{f_{t,d}}{|d|}$$

- A single t with $p(t|M_d) = 0$ will make $p(q|M_d) = \prod p(t|M_d)$ zero.

- This can be smoothed with the prior knowledge we have about the collection.

Smoothing

- Key intuition: A non-occurring term is possible (even though it didn't occur), . . .

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

- The maximum likelihood language model M_C^{ml} based on the term frequencies in the collection as a whole:

$$p\left(t \middle| M_C^{ml}\right) = \frac{l_t}{l_C}$$

  - l_t is the number of times the term shows up in the collection
  - l_C is the number of terms in the whole collection.

- We will use $p\left(t \middle| M_C^{ml}\right)$ to "smooth" $p(t|d)$ away from zero.

LM with Jelineck-Mercer smoothing

- The first approach we can do is to create a mixture model with both distributions:

$$p(q|d, C) = \lambda \cdot p(q|M_d) + (1 - \lambda) \cdot p(q|M_c)$$

- Mixes the probability from the document with the general collection frequency of the word.

- 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.

Mixture model: Summary

- What we model: *The user has some background knowledge about the collection and has a “document in mind” and generates the query from this document*.

$$p(q|d,C) \approx \prod_{t \in \{q \cap d\}} \big(\lambda \cdot p(t|M_d) + (1-\lambda) \cdot p(t|M_c)\big)$$

- The LMJM represents a **mixture of document probabilities and collection probabilities**.

What if we mix the frequencies instead? -> LM Dirichlet Smoothing

LM with Dirichlet smoothing

- We can use the prior knowledge about the mean of each term.

- The mean of the term in the collection should be our starting point when computing the term average on a document:
  - Imagine that we can add a fractional number occurrences to each term frequency.
  - Add $\mu = 1000$ occurrences of terms to a document according to the collection distribution.
  - The frequency of each term t_i would increase $\mu \cdot p(t|M_c)$
  - The length of each document increases by 1000.

- This will change the way we compute the mean of a term on a document.

Dirichlet smoothing

- We end up with the maximum a posteriori estimate of the term average:

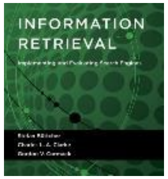
$$p\left(t \middle| M_d^{MAP}\right) = \frac{f_{t,d} + \mu \cdot p(t|M_c)}{|d| + \mu}$$

  - This is equivalent to using a Dirichlet prior with appropriate parameters.

- The ranking function becomes:

$$p(q|d) = \prod_{t \in q} \left(\frac{f_{t,d} + \mu \cdot p(t|M_c)}{|d| + \mu} \right)^{q_t}$$

Experimental comparison



	TREC45				Gov2			
	1998		1999		2005		2006	
Method	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**	**P@10**	**MAP**
Cosine TF-IDF	0.264	0.126	0.252	0.135	0.120	0.060	0.194	0.092
Proximity	0.396	0.124	0.370	0.146	0.425	0.173	0.562	0.23
No length norm. (rawTF)	0.266	0.106	0.240	0.120	0.298	0.093	0.282	0.097
D: rawTF+ noIDF Q: IDF	0.342	0.132	0.328	0.154	0.400	0.144	0.466	0.151
BM25	0.424	0.178	**0.440**	0.205	0.471	0.243	0.534	0.277
LMJM	0.390	0.179	0.432	0.209	0.416	0.211	0.494	0.257
LMD	**0.450**	**0.193**	0.428	**0.226**	**0.484**	**0.244**	**0.580**	**0.293**
BM25 + PRF	0.452	0.239	0.454	0.249	0.567	0.277	0.588	0.314
RRF	0.462	0.215	0.464	0.252	0.543	0.297	0.570	0.352
LR			0.446	0.266			0.588	0.309
RankSVM			0.420	0.234			0.556	0.268

Experimental comparison

- For long queries, the Jelinek-Mercer smoothing performs better than the Dirichlet smoothing.
- For short queries, the Dirichlet smoothing performs better than the Jelinek-Mercer smoothing.

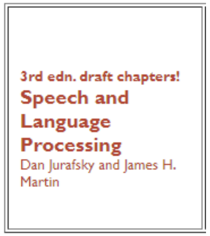
Method	Query	AP	Prec@10	Prec@20
LMJM	Title	0.227	0.323	0.265
LMD	Title	**0.256**	**0.352**	**0.289**
LMJM	Long	**0.280**	**0.388**	**0.315**
LMD	Long	0.279	0.373	0.303

Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.* 22, 2 (April 2004), 179-214.

Summary

Chapter 6

6.1 to 6.5

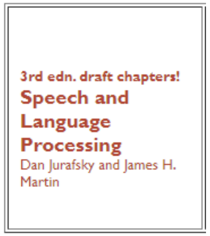


- Vector Space Model

- Language Models
  - Jelinek-Mercer smoothing
  - Dirichlet smoothing

Chapter 12

3.1 N-gram LM
3.5 Smoothing



- Both LM models need to calibrate one single parameter from the whole collection
  - (although there are known values that work well).

LMJM implementation

Initialize CountVectorizer

- Compute the term freq in all corpus
  - sum the colums of the count matrix

numpy sum with axis

- Compute doc length
  - sum the rows of the count matrix

numpy sum with axis

- Compute the term prob in all corpus : p(t|Mc)
  - Divide sum of columns of Mc by the sum of all terms

numpy reshape
element-wise division

- Compute the term prob in the document: p(t|Md)
  - Divide the Mc by the sum of rows of Mc

Division

Recap of basic probability

Information Retrieval

Recall a few probability basics

- For events A and B, the Bayes' Rule is:

$$p(A,B) = p(A|B)p(B) = p(B|A)p(A) \quad \text{(chain rule)}$$

$$p(A|B) = \frac{p(A,B)}{p(B)} = \frac{p(A)p(B|A)}{p(B)} \quad \text{(Bayes' rule)}$$

- Interpretation:

$$posterior = \frac{prior \cdot likelihood}{evidence} \Leftrightarrow p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

Recall a few probability basics

- Independence assumption:

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)} = \frac{p(A)\prod_i p(b_i|A)}{\prod_i p(b_i)}$$

- Odds: $O(A) = \frac{p(A)}{p(\bar{A})} = \frac{p(A)}{1 - p(A)}$

$$O(A|B) = \frac{p(A|B)}{p(\bar{A}|B)} = \frac{\frac{p(A)p(B|A)}{p(B)}}{\frac{p(\bar{A})p(B|\bar{A})}{p(B)}} = \frac{p(A)p(B|A)}{p(\bar{A})p(B|\bar{A})}$$

Recall a few probability basics

$$p(A|data) = \frac{p(A)p(data|A)}{p(data)}$$

$$p(SLB = campeão|data) = \frac{p(SLB = campeão)p(data|SLB = campeão)}{p(data)}$$

$$aposteriori = \frac{apriori \cdot verosimilhança}{evidencia}$$