Introduction to Natural Language Processing

a course taught as B4M36NLP at Open Informatics

by members of the Institute of Formal and Applied Linguistics

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Today: **Week 8, lecture**
Today's topic: **Probabilistic Models for Information Retrieval**
Today's teacher: **Pavel Pecina**

E-mail: pecina@ufal.mff.cuni.cz
WWW: http://ufal.mff.cuni.cz/∼pecina/

Contents

Introduction

Probability Ranking Principle

Binary Independence Model

Okapi BM25

Language models

Summary

Probabilistic IR models at a glance

- Classical probabilistic retrieval model
  1. Probability Ranking Principle
  2. Binary Independence Model
  3. BestMatch25 (Okapi)
- Language model approach to IR

Introduction

Probabilistic model vs. other models

Boolean model:

- Probabilistic models support ranking and thus are better than the simple Boolean model.

Vector space model:

- The vector space model is also a formally defined model that supports ranking.
- Why would we want to look for an alternative to the vector space model?

Probabilistic vs. vector space model

- Vector space model: rank documents according to similarity to query.
- The notion of similarity does not translate directly into an assessment of "*is the document a good document to give to the user or not?*"
- The most similar document can be highly relevant or completely nonrelevant.
- Probability theory is arguably a cleaner formalization of what we really want an IR system to do: give relevant documents to the user.

Basic Probability Theory

- For events A and B:
  - Joint probability $P(A \cap B)$: both events occurring
  - Conditional probability $P(A|B)$: A occurring given B has occurred
- Chain rule gives relationship between joint/conditional probabilities:

$$P(AB) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

- Similarly for the complement of an event $P(\overline{A})$:

$$P(\overline{A}B) = P(B|\overline{A})P(\overline{A})$$

- Partition rule: if B can be divided into an exhaustive set of disjoint subcases, then $P(B)$ is the sum of the probabilities of the subcases. A special case of this rule gives:

$$P(B) = P(AB) + P(\overline{A}B)$$

Basic probability theory cont'd

- Bayes' Rule for inverting conditional probabilities:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)}{\sum_{X\in\{A,\overline{A}\}} P(B|X)P(X)} \cdot P(A)$$

- Can be thought of as a way of updating probabilities:
  - Start off with prior probability $P(A)$ (initial estimate of how likely event A is in the absence of any other information).
  - Derive a posterior probability $P(A|B)$ after having seen the evidence B, based on the likelihood of B occurring in the two cases that A does or does not hold.

- Odds of an event is a kind of multiplier for how probabilities change:

$$\text{Odds:} \qquad O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1 - P(A)}$$

Probability Ranking Principle

The document ranking problem

- Ranked retrieval setup: given a collection of documents, the user issues a query, and an ordered list of documents is returned.
- Assume binary relevance: $R_{d,q}$ is a random dichotomous variable:

  $R_{d,q} = 1$ if document d is relevant w.r.t query q

  $R_{d,q} = 0$ otherwise.

- Often we write just R for $R_{d,q}$
- Probabilistic ranking orders documents decreasingly by their estimated probability of relevance w.r.t. query: $P(R = 1|d, q)$.
- Assume that the relevance of each document is independent of the relevance of other documents.

Probability Ranking Principle (PRP)

PRP in brief:

- If the retrieved documents (w.r.t a query) are ranked decreasingly on their probability of relevance, then the effectiveness of the system will be the best that is obtainable.

PRP in full:

- If [the IR] system's response to each [query] is a ranking of the documents [...] in order of decreasing probability of relevance to the [query], where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

Binary Independence Model

Binary Independence Model (BIM)

Traditionally used with the PRP, with the following assumptions:

1. 'Binary' (equivalent to Boolean): documents and queries represented as binary term incidence vectors.
   - E.g., document d represented by vector $\vec{x} = (x_1, \ldots, x_M)$, where $x_t = 1$ if term t occurs in d and $x_t = 0$ otherwise.
   - Different documents may have the same vector representation.
   - Similarly, we represent q by the incidence vector $\vec{q}$.
2. 'Independence': no association between terms (not true, but practically works – 'naive' assumption of Naive Bayes models).

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

Binary Independence Model (1)

To make a probabilistic retrieval strategy precise, need to estimate how terms in documents contribute to relevance:

- Find measurable statistics (term frequency, document frequency, document length) that affect judgments about document relevance.
- Combine these statistics to estimate the probability $P(R|d, q)$ of document relevance.
- Next: how exactly we can do this.

Binary Independence Model (2)

$P(R|d,q)$ is modeled using term incidence vectors as $P(R|\vec{x},\vec{q})$:

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

$$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}$$

- $P(\vec{x}|R=1,\vec{q})$ and $P(\vec{x}|R=0,\vec{q})$: probability that if a relevant or nonrelevant document is retrieved, then that document's representation is $\vec{x}$.
- Use statistics about the document collection to estimate these probabilities.

Binary Independence Model (3)

$P(R|d,q)$ is modeled using term incidence vectors as $P(R|\vec{x},\vec{q})$:

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

$$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}$$

- $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$: prior probability of retrieving a relevant or nonrelevant document for a query $\vec{q}$.
- Estimate $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$ from percentage of relevant documents in the collection.
- Since a document is either relevant or nonrelevant to a query, we must have that: $P(R=1|\vec{x},\vec{q}) + P(R=0|\vec{x},\vec{q}) = 1$

Deriving a ranking function for query terms (1)

- Given a query q, ranking documents by $P(R=1|d,q)$ is modeled under BIM as ranking them by $P(R=1|\vec{x},\vec{q})$.
- Easier: rank documents by their odds of relevance (same ranking):

$$O(R|\vec{x},\vec{q}) = \frac{P(R=1|\vec{x},\vec{q})}{P(R=0|\vec{x},\vec{q})} = \frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}} =$$

$$= \frac{P(R=1|\vec{q})}{P(R=0|\vec{q})} \cdot \frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})}$$

- $\frac{P(R=1|\vec{q})}{P(R=0|\vec{q})} = O(R|\vec{q})$ is a constant for a given query $\rightarrow$ can be ignored.

Deriving a ranking function for query terms (2)

At this point we make the Naive Bayes conditional independence assumption that the presence/absence of a word in a document is independent of the presence/absence of any other word (given the query):

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} = \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}$$

So:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \cdot \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}$$

Exercise

Naive Bayes conditional independence assumption: the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query).

Why is this wrong? Good example?

PRP assumes that the relevance of each document is independent of the relevance of other documents.

Why is this wrong? Good example?

Deriving a ranking function for query terms (3)

- Since each x_t is either 0 or 1, we can separate the terms:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \cdot \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} =$$

$$= O(R|\vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})}$$

Deriving a ranking function for query terms (4)

- Let $p_t = P(x_t = 1|R = 1, \vec{q})$ be the probability of a term appearing in relevant document.
- Let $u_t = P(x_t = 1|R = 0, \vec{q})$ be the probability of a term appearing in a nonrelevant document.
- Can be displayed as a contingency table:

	document	relevant ($R = 1$)	nonrelevant ($R = 0$)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

Deriving a ranking function for query terms (5)

- Additional simplifying assumption: terms not occurring in the query are equally likely to occur in relevant and nonrelevant documents.

  $\rightarrow$ If $q_t = 0$, then $p_t = u_t$.

- We only consider terms in the products that appear in the query:

$$O(R|\vec{x}, \vec{q}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}$$

- The left product is over query terms found in the document and the right product is over query terms not found in the document.

Deriving a ranking function for query terms (6)

- Including the query terms found in the document into the right product, but simultaneously dividing by them in the left product:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t:q_t=1} \frac{1-p_t}{1-u_t}$$

- The left product is still over query terms found in the document, but the right product is now over all query terms, hence constant for a particular query and can be ignored.

→ The only quantity that needs to be estimated to rank documents w.r.t a query is the left product.

- We can equally rank documents by the logarithm of this term, since log is a monotonic function.
- Hence the Retrieval Status Value (RSV) in this model:

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

Deriving a ranking function for query terms (7)

Equivalent: rank documents using log odds ratios for the query terms c_t:

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} - \log \frac{u_t}{1-u_t}$$

- The odds ratio is the ratio of two odds: (i) the odds of the term appearing if the document is relevant ($p_t/(1-p_t)$), and (ii) the odds of the term appearing if the document is nonrelevant ($u_t/(1-u_t)$)
- $c_t = 0$: term has equal odds of appearing in relevant and nonrel. docs.
- c_t positive: higher odds to appear in relevant documents.
- c_t negative: higher odds to appear in nonrelevant documents.

Term weight c_t in BIM

- $c_t = \log \frac{p_t}{(1-p_t)} - \log \frac{u_t}{1-u_t}$ functions as a term weight.
- Retrieval status value for document d: $RSV_d = \sum_{x_t=q_t=1} c_t$.
- So BIM and vector space model are identical on an operational level

  ... except that the term weights are different.
- In particular: we can use the same data structures (inverted index etc) for the two models.

How to compute probability estimates

For each term t in a query, estimate c_t in the whole collection using a contingency table of counts of documents in the collection, where df_t is the number of documents that contain term t:

	documents	relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

$$p_t = s/S$$

$$u_t = (\mathrm{df}_t - s)/(N - S)$$

$$c_t = K(N, \mathrm{df}_t, S, s) = \log \frac{s/(S-s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S - s))}$$

Avoiding zeros

- If any of the counts is a zero, then the term weight is not well-defined.
- Maximum likelihood estimates do not work for rare events.
- To avoid zeros: add 0.5 to each count (Expected Likelihood Estimation).
- For example, use $S - s + 0.5$ in formula for $S - s$.

Simplifying assumption

- Assuming that relevant documents are a very small percentage of the collection, approximate statistics for nonrelevant documents by statistics from the whole collection.
- Hence, u_t (the probability of term occurrence in nonrelevant documents for a query) is df_t/N and

$$\log[(1-u_t)/u_t] = \log[(N-\mathrm{df}_t)/\mathrm{df}_t] \approx \log N/\mathrm{df}_t$$

- This should look familiar to you ...
- The above approximation cannot easily be extended to relevant documents.

Probability estimates in relevance feedback

- Statistics of relevant documents (p_t) in relevance feedback can be estimated using maximum likelihood estimation or expected likelihood estimation (add 0.5).
  - Use the frequency of term occurrence in known relevant documents.
- This is the basis of probabilistic approaches to relevance feedback weighting in a feedback loop.

Probability estimates in ad-hoc retrieval

- Ad-hoc retrieval: no user-supplied relevance judgments available.
- In this case: assume that p_t is constant over all terms x_t in the query and that $p_t = 0.5$.
- Each term is equally likely to occur in a relevant document, and so the p_t and $(1 - p_t)$ factors cancel out in the expression for *RSV*.
- Weak estimate, but doesn't disagree violently with expectation that query terms appear in many but not all relevant documents.
- Combining this method with the earlier approximation for u_t, the document ranking is determined simply by which query terms occur in documents scaled by their idf weighting.
- For short documents (titles or abstracts) in one-pass retrieval situations, this estimate can be quite satisfactory.

History and summary of assumptions

- Among the oldest formal models in IR:
  - Maron & Kuhns, 1960: Since an IR system cannot predict with certainty which document is relevant, we should deal with probabilities.

- Assumptions for getting reasonable approximations of the needed probabilities (in the BIM):
  1. Boolean representation of documents/queries/relevance
  2. Term independence
  3. Out-of-query terms do not affect retrieval
  4. Document relevance values are independent

How different are vector space model and BIM?

- They are not that different.
- In either case you build an information retrieval scheme in the exact same way.
- For probabilistic IR, at the end, you score queries not by cosine similarity and tf-idf in a vector space, but by a slightly different formula motivated by probability theory.
- Next: how to add term frequency and length normalization to the probabilistic model.

Okapi BM25

Okapi BM25: Overview

- Okapi BM25 is a probabilistic model that incorporates term frequency (i.e., it's nonbinary) and length normalization.
- BIM was originally designed for short catalog records of fairly consistent length, and it works reasonably in these contexts.
- For modern full-text search collections, a model should pay attention to term frequency and document length.
- BestMatch25 (a.k.a BM25 or Okapi) is sensitive to these quantities.
- BM25 is one of the most widely used and robust retrieval models.

Okapi BM25: Starting point

The simplest score for document d is just idf weighting of the query terms present in the document:

$$RSV_d = \sum_{t \in q} \log \frac{N}{\mathrm{df}_t}$$

Okapi BM25 basic weighting

Improve idf term by factoring in term frequency and document length.

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b\,(L_d/L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

- tf_{td}: term frequency in document d
- L_d: length of document d
- L_{ave}: average document length in the whole collection
- k_1: tuning parameter controlling document term frequency scaling
- b: tuning parameter controlling scaling by document length

Exercise

Okapi BM25:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b\,(L_d/L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

1. Interpret BM25 weighting formula for $k_1 = 0$
2. Interpret BM25 weighting formula for $k_1 = 1$ and $b = 0$
3. Interpret BM25 weighting formula for $k_1 \mapsto \infty$ and $b = 0$
4. Interpret BM25 weighting formula for $k_1 \mapsto \infty$ and $b = 1$

Okapi BM25 weighting for long queries

- For long queries, use similar weighting for query terms:

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b\,(L_d/L_{\mathrm{ave}})) + \mathrm{tf}_{td}} \cdot \frac{(k_3 + 1)\mathrm{tf}_{tq}}{k_3 + \mathrm{tf}_{tq}}$$

- tf_{tq}: term frequency in the query q
- k_3: tuning parameter controlling term frequency scaling of the query
- No length normalization of queries (because retrieval is being done with respect to a single fixed query)
- The above tuning parameters should ideally be set to optimize performance on a development test collection. In the absence of such optimization, experiments have shown reasonable values are to set k_1 and k_3 to a value between 1.2 and 2 and $b = 0.75$

Which ranking model should I use?

- I want something basic and simple → use vector space with tf-idf weighting.
- I want to use a state-of-the-art ranking model with excellent performance → use language models or BM25 with tuned parameters.
- In between: BM25 or language models with no or just one tuned parameter.

Language models

Using language models for Information Retrieval

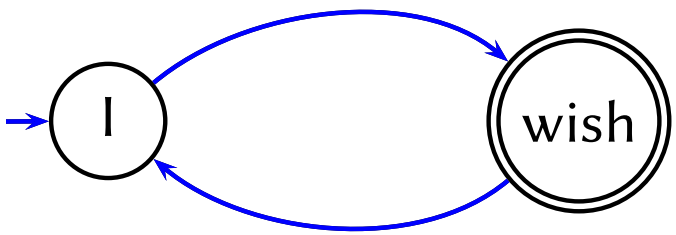
View the document d as a generative model that generates the query q.

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.

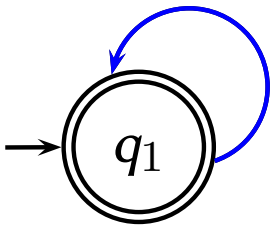


I wish I wish I wish I wish ...

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

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

  - $|d|$: length of d
  - $\mathrm{tf}_{t,d}$: # 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.

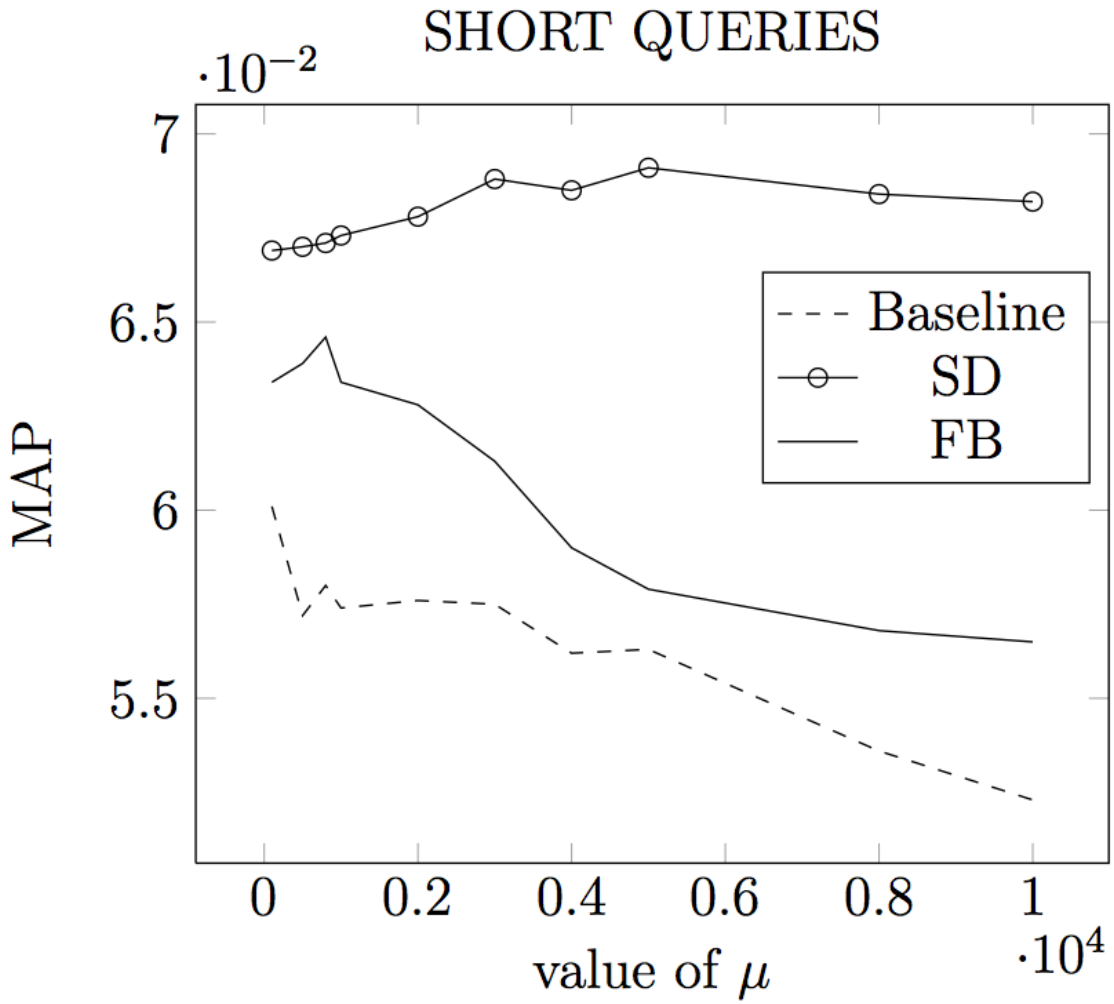
$$\hat{P}(t|d) = \frac{\text{tf}_{t,d} + \mu \hat{P}(t|M_c)}{L_d + \mu}$$

- 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.

Summary

What we have learned

1. What is IR
   - information need, query, ranking
2. Text processing
   - tokenization, mormalization, lemmatization, stemming
3. Boolean model
   - queries as boolean expression, inverted index
4. Vector space model
   - vector representations, cosine similarity, TF-IDF weighting
5. Evaluation in IR
   - Precision, Recall, F-measure, P/R curves, Mean average precision
6. Binary independence model
   - Probability Ranking Principle, Okapi BM25
7. Language models for IR
   - $p(q|d)$

What we have't

1. phrase queries, proximity queries
2. spelling correction
3. inverted index construction, index compression
4. relevance feedback, query expansion
5. summary construction
6. text clasification
7. document clustering
8. latent semantic indexing
9. web-search
10. ...