Language Models

Northeastern University
College of Computer and Information Science

CS6200: Information Retrieval

Slides by: Jesse Anderton

What's wrong with VSMs?

Vector Space Models work reasonably well, but have a few problems:

- They are based on bag-of-words, so they ignore grammatical context and suffer from term mismatch.
- They don't adapt to the user or collection, but ideal term weights are user- and domain-specific.
- They are heuristic-based, and don't have much explanatory power.

Probabilistic Modeling

We can address these problems by moving to probabilistic models, such as language models:

- We can take grammatical context into account, and trade off between using more context and performing faster inference.
- The model can be trained from a particular collection, or conditioned based on user- and domain-specific features.
- The model is interpretable, and makes concrete predictions about query and document relevance.

In this Module…

1. Ranking as a probabilistic classification task
2. Some specific probabilistic models for classification
3. Smoothing: estimating model parameters from sparse data
4. A probabilistic approach to pseudo-relevance feedback

Ranking with Probabilistic Models

Imagine we have a function that gives us the probability that a document D is relevant to a query Q, $P(R=1|D, Q)$. We call this function a *probabilistic model*, and can rank documents by decreasing probability of relevance.

There are many useful models, which differ by things like:

- Sensitivity to different document properties, like grammatical context
- Amount of training data needed to train the model parameters
- Ability to handle noise in document data or relevance labels

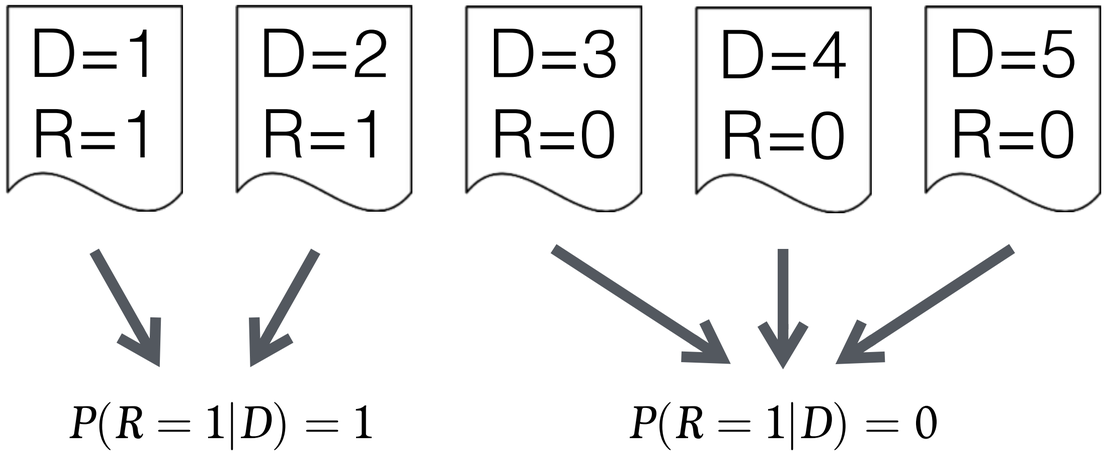
For simplicity here, we will hold the query constant and consider $P(R=1|D)$.

The Flaw in our Plan

Suppose we have documents and relevance labels, and we want to empirically measure $P(R=1|D)$.

Each document has only one relevance label, so every probability is either 0 or 1. Worse, there is no way to generalize to new documents.

Instead, we estimate the probability of documents given relevance labels, $P(D|R=1)$.



	$D=1$	$D=2$	$D=3$	$D=4$	$D=5$
$P(D\|R=1)$	1/2	1/2	0	0	0
$P(D\|R=0)$	0	0	1/3	1/3	1/3

language model for doc = prob dist over words/terms

Bayes' Rule

We can estimate $P(D|R{=}1)$, not $P(R{=}1|D)$, so we apply Bayes' Rule to estimate document relevance.

- $P(D|R{=}1)$ gives the probability that a relevant document would have the properties encoded by the random variable D.
- $P(R{=}1)$ is the probability that a randomly-selected document is relevant.

$$P(R=1|D) = \frac{P(D|R=1)P(R=1)}{P(D)}$$

$$= \frac{P(D|R=1)P(R=1)}{\sum_r P(D|R=r)P(R=r)}$$

Bayesian Classification

Starting from Bayes' Rule, we can easily build a classifier to tell us whether documents are relevant. We will say a document is relevant if:

want prob(relevance | doc)

$$P(R=1|D) > P(R=0|D)$$

$$\Longrightarrow \frac{P(D|R=1)P(R=1)}{P(D)} > \frac{P(D|R=0)P(R=0)}{P(D)}$$

$$\Longrightarrow \frac{P(D|R=1)}{P(D|R=0)} > \frac{P(R=0)}{P(R=1)}$$

We can estimate $P(D|R{=}1)$ and $P(D|R{=}0)$ using a language model, and $P(R{=}0)$ and $P(R{=}1)$ based on the query, or using a constant. Note that for large web collections, $P(R{=}1)$ is very small for virtually any query.

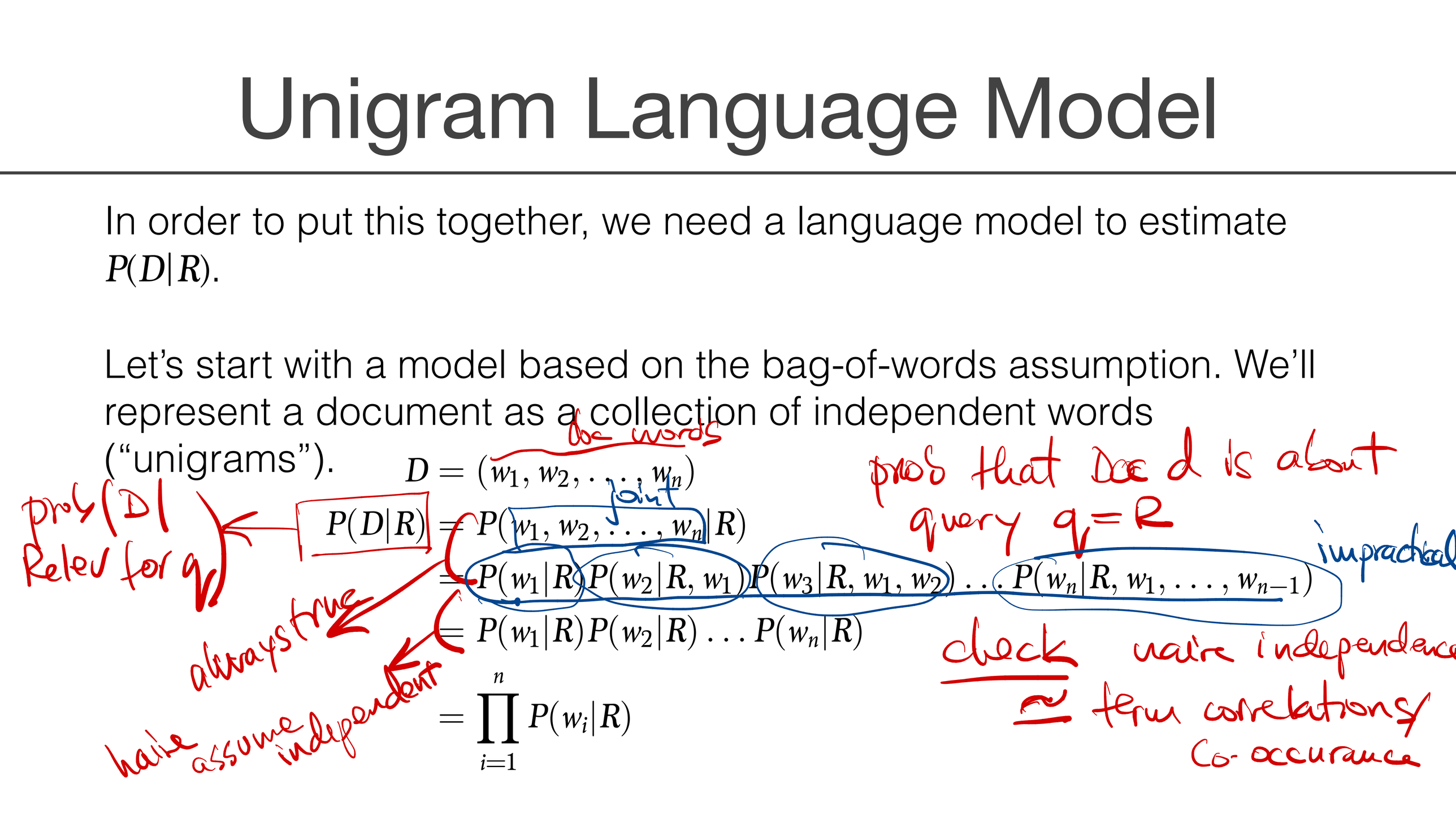
R = relevance = relevance for a particular query.

Unigram Language Model

In order to put this together, we need a language model to estimate $P(D|R)$.

Let's start with a model based on the bag-of-words assumption. We'll represent a document as a collection of independent words ("unigrams").

$$
\begin{aligned}
D &= (w_1, w_2, \ldots, w_n) \\
P(D|R) &= P(w_1, w_2, \ldots, w_n|R) \\
&= P(w_1|R)P(w_2|R, w_1)P(w_3|R, w_1, w_2)\ldots P(w_n|R, w_1, \ldots, w_{n-1}) \\
&= P(w_1|R)P(w_2|R)\ldots P(w_n|R) \\
&= \prod_{i=1}^{n} P(w_i|R)
\end{aligned}
$$



Example

Let's consider querying a collection of five short documents with a simplified vocabulary: the only words are apple, baker, and crab.

Document	Rel?	apple?	baker?	crab?
apple apple crab	1	1	0	1
crab baker crab	0	0	1	1
apple baker baker	1	1	1	0
crab crab apple	0	1	0	1
baker baker crab	0	0	1	1

Term	# Rel	# Non Rel	$P(w\|R=1)$	$P(w\|R=0)$
apple	2	1	2/2	1/3
baker	1	2	1/2	2/3
crab	1	3	1/2	3/3

$$P(R=1) = 2/5$$

$$P(R=0) = 3/5$$

Example

Is “apple baker crab” relevant?

$$\frac{P(D|R=1)}{P(D|R=0)} \stackrel{?}{>} \frac{P(R=0)}{P(R=1)}$$

$$\frac{\prod_i P(w_i|R=1)}{\prod_i P(w_i|R=0)} \stackrel{?}{>} \frac{P(R=0)}{P(R=1)}$$

$$\frac{P(apple=1|R=1)P(baker=1|R=1)P(crab=1|R=1)}{P(apple=1|R=0)P(baker=1|R=0)P(crab=1|R=0)} \stackrel{?}{>} \frac{0.6}{0.4}$$

$$\frac{1 \cdot 0.5 \cdot 0.5}{0.\bar{3} \cdot 0.\bar{6} \cdot 1} \stackrel{?}{>} \frac{0.6}{0.4}$$

$$1.125 < 1.5$$

Term	$P(w\|R=1)$	$P(w\|R=0)$
apple	1	1/3
baker	1/2	2/3
crab	1/2	1

$$P(R=1) = 2/5$$

$$P(R=0) = 3/5$$

Retrieval With Language Models

So far, we've focused on language models like $P(D = w_1, w_2, \ldots, w_n)$. Where's the query?

Remember the key insight from vector space models: we want to represent queries and documents in the same way. The query is just a "short document:" a sequence of words. There are three obvious approaches we can use for ranking:

1. Query likelihood: Train a language model on a document, and estimate the query's probability.

2. Document likelihood: Train a language model on the query, and estimate the document's probability.

3. Model divergence: Train language models on the document and the query, and compare them.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

ranking A-s for fix B

$P(A|B) \overset{\text{rank}}{\sim} P(B|A) \cdot P(A)$ (proportion ✓, rank ✓)

NOT EXACT, not a prob

Query Likelihood Retrieval

Suppose that the query specifies a topic. We want to know the probability of a document being generated from that topic, or $P(D|Q)$.

However, the query is very small, and documents are long: document language models have less variance.

In the *Query Likelihood Model*, we use Bayes' Rule to rank documents based on the probability of generating the query from the documents' language models.

$$P(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

Assuming uniform prior

$$= P(Q|D)$$

$$= \prod_{w \in Q} P(w|D)$$

Naive Bayes unigram model

$$\overset{rank}{=} \sum_{w \in Q} \log P(w|D)$$

Numerically stable version

$$\text{score}(d) \overset{rank}{=} \sum_{w \in Q} \log P(w|d)$$

pseudocode prototype

init: score(d)=0 ∀d

For w ∈ q

  For each doc d ∈ invlist(w)

    score(d) = score(d) + match(w,d)

    skip doc [d ∌ w ?] incorrect

term w does not appear in doc d ⇒ penalty ⇒ log[P(w|d)] (small) ⇒ deep negative

- can I implement Lang. Mode Query Likelihood this way?

  - problem: match ∈ [log(prob) ∈ (-∞, 0]]

    max match value is 0 = log(1)

    max prob = 1

if w ∉ d ⇒ match(w,d) signif negative ⇒ we cannot simply skip doc d.

Solution for L.M. change init function

init score(d) = ?

Example: Query Likelihood

Wikipedia: WWI
World War I (**WWI** or **WW1** or **World War One**), also known as the **First World War** or the **Great War**, was a global war centred in Europe that began on 28 July 1914 and lasted until 11 November 1918. More than 9 million combatants and 7 million civilians died as a result of the war, a casualty rate exacerbated by the belligerents' technological and industrial sophistication, and tactical stalemate. It was one of the deadliest conflicts in history, paving the way for major political changes, including revolutions in many of the nations involved.

Query: "deadliest war in history"

Term	P(w|D)	log P(w|D)
deadliest	1/94 = 0.011	-1.973
war	6/94 = 0.063	-1.195
in	3/94 = 0.032	-1.496
history	1/94 = 0.011	-1.973
	Π = 2.30e-7	**Σ = -6.637**

Keyword based query 34 terms
=> Q.L. model

Example: Query Likelihood

Wikipedia: WWI
World War I (**WWI** or **WW1** or **World War One**), also known as the **First World War** or the **Great War**, was a global war centred in Europe that began on 28 July 1914 and lasted until 11 November 1918. More than 9 million combatants and 7 million civilians died as a result of the war, a casualty rate exacerbated by the belligerents' technological and industrial sophistication, and tactical stalemate. It was one of the deadliest conflicts in history, paving the way for major political changes, including revolutions in many of the nations involved.

Query: “deadliest war in history”

Term	P(w|D)	log P(w|D)
deadliest	1/94 = 0.011	-1.973
war	6/94 = 0.063	-1.195
in	3/94 = 0.032	-1.496
history	1/94 = 0.011	-1.973
	Π = 2.30e-7	**Σ = -6.637**

Example: Query Likelihood

Wikipedia: Taiping Rebellion

The **Taiping Rebellion** was a massive civil war in southern China from 1850 to 1864, against the ruling Manchu Qing dynasty. It was a millenarian movement led by Hong Xiuquan, who announced that he had received visions, in which he learned that he was the younger brother of Jesus. At least 20 million people died, mainly civilians, in one of the deadliest military conflicts in history.

Query: "deadliest war in history"

Term	P(w|D)	log P(w|D)
deadliest	1/56 = 0.017	-1.748
war	1/56 = 0.017	-1.748
in	2/56 = 0.035	-1.447
history	1/56 = 0.017	-1.748
	Π = 2.56e-8	**Σ = –6.691**

Summary: Language Model

There are many ways to move beyond this basic model.

- Use n-gram or skip-gram probabilities, instead of unigrams.
- Model document probabilities $P(D)$ based on length, authority, genre, etc. instead of assuming a uniform probability.
- Use the tools from the VSM slides: stemming, stopping, etc.

Next, we'll see how to fix a major issue with our probability estimates: what happens if a query term doesn't appear in the document?

Retrieval With Language Models

There are three obvious ways to perform retrieval using language models:

1. **Query Likelihood Retrieval** trains a model on the document and estimates the query's likelihood. We've focused on these so far.
2. **Document Likelihood Retrieval** trains a model on the query and estimates the document's likelihood. Queries are very short, so these seem less promising.
3. **Model Divergence Retrieval** trains models on both the document and the query, and compares them.

HW1

→ not good except in special cases.

→ used a lot in practice (not req HW1)

Comparing Distributions

The most common way to compare probability distributions is with *Kullback-Liebler ("KL") Divergence*.

measures similarity (closeness) between 2 distributions

$E_p(V) = \sum_{x \in \Omega} p(x) \cdot V(x)$

weighted by p

This is a measure from Information Theory which can be interpreted as the expected number of bits you would waste if you compressed data distributed along p as if it was distributed along q.

If $p = q$, $D_{KL}(p||q) = 0$.

$$D_{KL}(p||q) = \sum_e p(e) \log \frac{p(e)}{q(e)}$$

$= E_p\left[\log \frac{p}{q}\right]$

bits required to express diff $\frac{p}{q}$

log = $\log_2$ unless specified

Divergence-based Retrieval

Model Divergence Retrieval works as follows:

1. Choose a language model for the query, $p(w|q)$.
2. Choose a language model for the document, $p(w|d)$.
3. Rank by $-D_{KL}(p(w|q) \,||\, p(w|d))$ – more divergence means a worse match.

This can be simplified to a cross-entropy calculation, as shown to the right.

$$
\begin{aligned}
& D_{KL}(p(w|q)||p(w|d)) \\
&= \sum_w p(w|q) \log \frac{p(w|q)}{p(w|d)} \\
&= \sum_w p(w|q) \log p(w|q) - \sum_w p(w|q) \log p(w|d) \\
&\overset{rank}{=} - \sum_w p(w|q) \log p(w|d)
\end{aligned}
$$

discrete

$P = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}$

$q = \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{2}$

$\Omega = \{x_1, x_2, x_3, x_4\}$

$$KL(p\|q) = \sum_{x \in \Omega} p(x) \cdot \log\left(\frac{p(x)}{q(x)}\right)$$

$$= \frac{1}{2} \cdot \log\left(\frac{1/2}{1/4}\right) + \frac{1}{4} \log\left(\frac{1/4}{1/8}\right) +$$

$$+ \frac{1}{8} \log\left(\frac{1/8}{1/8}\right) + \frac{1}{8} \log\left(\frac{1/8}{1/2}\right) =$$

$$= \frac{1}{2} \cdot \underbrace{\log(2)}_{1} + \frac{1}{4} \underbrace{\log(2)}_{1} + \frac{1}{8} \underbrace{\log(1)}_{0} + \frac{1}{8} \underbrace{\log\left(\frac{1/8}{1/2}\right)}_{-2}$$

continuous $\Sigma \to \int$

Exercise

- $KL(p\|q) \geq 0$
- $\min KL(p\|q) = 0 \iff p(x) = q(x) \ \forall x$
- KL NOT SYMMETRIC (as usually distance)

$$KL(p\|q) \neq KL(q\|p)$$

Retrieval Flexibility

Model Divergence Retrieval generalizes the Query and Document Likelihood models, and is the most flexible of the three.

Any language model can be used for the query or document. They don't have to be the same. It can help to smooth or normalize them differently.

If you pick the maximum likelihood model for the query, this is equivalent to the query likelihood model.

$$\text{Pick } p(w|q) := \frac{tf_{w,q}}{|q|} = \frac{1}{|q|}$$

$$D_{KL}(p(w|q)||p(w|d)) \overset{rank}{=} -\sum_w p(w|q) \log p(w|d)$$

$$= -\sum_w \frac{1}{|q|} \log p(w|d)$$

Equivalence to Query Likelihood Model

Example: Model Divergence Retrieval

We make the following model choices:

- $p(w|q)$ is Dirichlet-smoothed with a background of words used in historical queries.
- $p(w|d)$ is Dirichlet-smoothed with a background of words used in documents from the corpus.
- $\Sigma_w\ qf_w$ = 500,000
- $\Sigma_w\ cf_w$ = 1,000,000,000

$$\text{Let } qf_w := count(\text{word } w \text{ in query log})$$

$$p(w|q, \mu = 2) = \frac{tf_{w,q} + 2 \times \frac{qf_w}{\sum_w qf_w}}{|q| + 2}$$

$$p(w|d, \mu = 2000) = \frac{tf_{w,d} + 2,000 \times \frac{cf_w}{\sum_w cf_w}}{|d| + 2,000}$$

$$D_{KL}(p(w|q)||p(w|d)) \overset{rank}{=} -\sum_w p(w|q) \log p(w|d)$$

$$= -\sum_w \frac{tf_{w,q} + 2 \times \frac{qf_w}{\sum_w qf_w}}{|q| + 2} \log \frac{tf_{w,d} + 2,000 \times \frac{cf_w}{\sum_w cf_w}}{|d| + 2,000}$$

Ranking by (negative) KL-Divergence provides a very flexible and theoretically-sound retrieval system.

Example: Model Divergence Retrieval

Wikipedia: WWI
World War I (**WWI** or **WW1** or **World War One**), also known as the **First World War** or the **Great War**, was a global war centred in Europe that began on 28 July 1914 and lasted until 11 November 1918. More than 9 million combatants and 7 million civilians died as a result of the war, a casualty rate exacerbated by the belligerents' technological and industrial sophistication, and tactical stalemate. It was one of the

$$\sum_w \frac{tf_{w,q} + 2 \times \frac{qf_w}{\sum_w qf_w}}{|q| + 2} \log \frac{tf_{w,d} + 2,000 \times \frac{cf_w}{\sum_w cf_w}}{|d| + 2,000}$$

Query: “world war one”

	qf	cf	p(w|q)	p(w|d)	Score
world	2,500	90,000	0.202	0.002	-1.891
war	2,000	35,000	0.202	0.003	-1.700
one	6,000	5E+07	0.205	0.049	-0.893
					-4.484

Example: Model Divergence Retrieval

$$\sum_w \frac{tf_{w,q} + 2 \times \frac{qf_w}{\sum_w qf_w}}{|q| + 2} \log \frac{tf_{w,d} + 2,000 \times \frac{cf_w}{\sum_w cf_w}}{|d| + 2,000}$$

Wikipedia: Taiping Rebellion

The **Taiping Rebellion** was a massive civil war in southern China from 1850 to 1864, against the ruling Manchu Qing dynasty. It was a millenarian movement led by Hong Xiuquan, who announced that he had received visions, in which he learned that he was the younger brother of Jesus. At least 20 million people died, mainly civilians, in one of

Query: "world war one"

	qf	cf	p(wlq)	p(wld)	Score
world	2,500	90,000	0.202	8.75E-05	-2.723
war	2,000	35,000	0.202	0.001	-2.199
one	6,000	5E+07	0.205	0.049	-0.890
					-5.812

Modeling Language

Although the bag of words model works very well for text classification, it is intuitively unsatisfying – it assumes the words in a document are independent, given the relevance label, and nobody believes this.

What could we replace it with?

- A “bag of paragraphs” wouldn’t work – too many paragraphs are unique in the collection, so we can’t do meaningful statistics without subdividing them.

- A “bag of sentences” is better, but not much – many sentences are unique, and two documents expressing the same thought are unlikely to choose exactly the same sentence. We need similar documents to have similar features.

- We’ll use sets of words, called *n-grams*, and consider sets of different sizes to balance between good probability estimates (for small n) and semantic nuance (for large n).

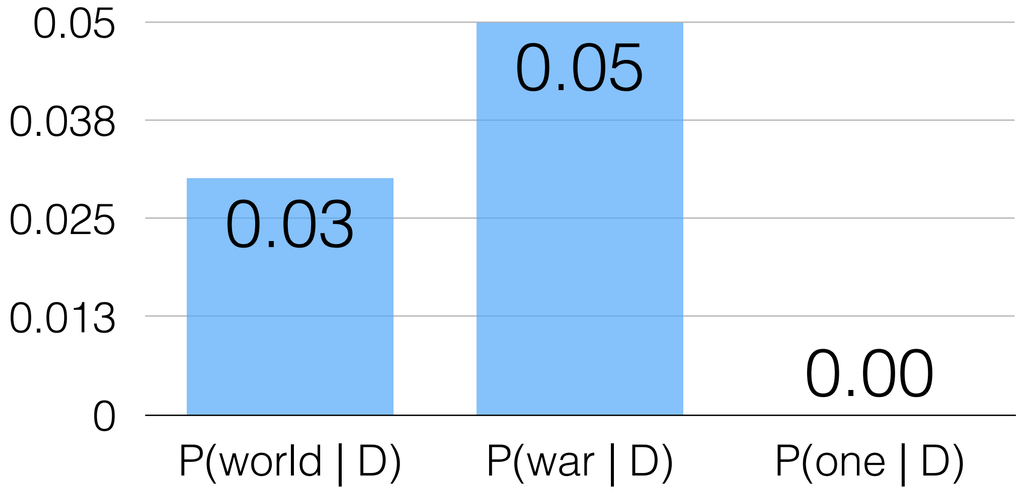
Probability Estimation

Maximum likelihood probability estimates assign zero probability to terms missing from the training data.

This is catastrophic for a Naive Bayes retrieval model: any document that doesn't contain all query terms will get a matching score of zero.

Many other probabilistic models have similar problems. Only truly impossible events should have zero probability.

Query: "world war one"



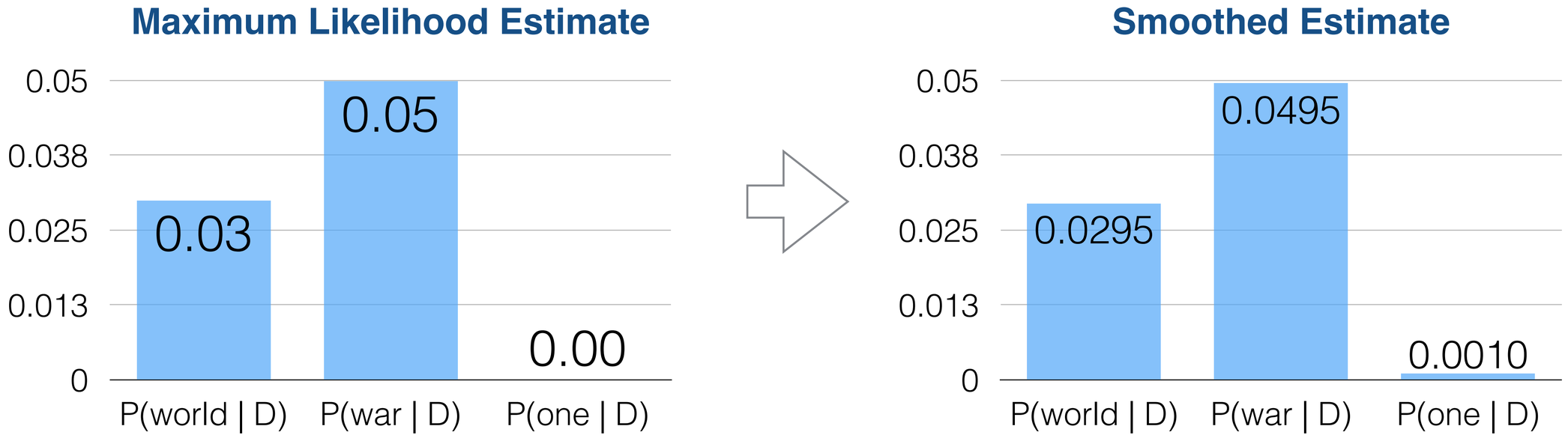
Query Likelihood Model

$$P(D|Q) \overset{rank}{=} \prod_{w \in Q} P(w|D)$$

$$= 0.03 \cdot 0.05 \cdot 0$$

Smoothing

The solution is to adjust our probability estimates by taking some probability away from the most-likely events, and moving it to the less-likely events.



This makes the probability distribution less spiky, or “smoother.” The probabilities all move just a little toward the mean.

Smoothing

Smoothing is important for many reasons.

- Assigning zero probability to possible events is incorrect.
- Maximum likelihood estimates from your data don't generalize perfectly to new data, so a Bayesian update from some kind of prior works better.

However, *uniform* smoothing doesn't work very well for language modeling. Next, we'll see why that is, and how we can do better.

Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval.

Laplace Smoothing

Laplace Smoothing, aka “add-one smoothing,” smooths maximum likelihood estimates by adding one count to each event.

$$P(e) = \frac{count(e) + 1}{\sum_{e \in events} (count(e) + 1)}$$

$$P(w|d) = \frac{tf_{w,d} + 1}{|d| + |V|}$$

This is equivalent to a Bayesian posterior with a uniform prior, as we'll see.

Pierre-Simon Laplace (1745-1827)

Image from Wikipedia

Deriving Laplace Smoothing

If we assume nothing about a document's vocabulary distribution, we will use uniform probabilities for all terms.

When we observe the terms in a document, the Bayesian update of these probabilities yields Laplace smoothing.

This Bayesian posterior is our smoothed estimate of the vocabulary distribution for the document's topic.

$$P(\pi|a) \text{ is } Dirichlet(\pi|a_1, \ldots, a_V) \propto \prod_{i=1}^{V} \pi_i^{a_i - 1}$$

$$P(d|\pi) \text{ is } Multinomial(\pi) \propto \prod_{i=1}^{V} \pi_i^{tf_{i,d}}$$

$$P(w|d) \propto P(d|\pi)P(\pi|a) = \prod_{i=1}^{V} \pi_i^{tf_{i,d} + a_i - 1}$$

$$\text{is } Dirichlet(\pi|a_1 + tf_{1,d}, \ldots, a_V + tf_{V,d})$$

$$\mathbb{E}[P(w|d)|a = 1] = \frac{tf_{1,d} + 1}{|d| + V}$$

Add-α Smoothing

Laplace smoothing can be generalized from add-one smoothing to add-α smoothing, for $\alpha \in (0, 1]$.

This lets you tune the amount of smoothing you want to use: smaller values of α are closer to the maximum likelihood estimate.

$$P(e) = \frac{count(e) + a}{\sum_{e \in events} (count(e) + a)}$$

$$P(w|d) = \frac{tf_{w,d} + a}{|d| + a|V|}$$

Limits of Uniform Smoothing

Uniform smoothing assigns the same probability to all unseen words, which isn't realistic. This is easiest to see for n-gram models:

$$P(house|the, white) > P(effortless|the, white)$$

We strongly believe that "house" is more likely to follow "the white" than "effortless" is, even if neither trigram appears in our training data.

Our bigram counts should help: "white house" probably appears more often than "white effortless." We can use bigram probabilities as a *background distribution* to help smooth our trigram probabilities.

Jelinek-Mercer Smoothing

One way to combine foreground and background distributions is to take their linear combination. This is the simplest form of Jelinek-Mercer Smoothing.

$$\hat{p}(e) = \lambda p_{fg}(e) + (1-\lambda) p_{bg}(e), 0 < \lambda < 1$$

For instance, you can smooth n-grams with (n-1)-gram probabilities.

$$\hat{p}(w_n|w_1,\ldots,w_{n-1}) = \lambda p(w_n|w_1,\ldots,w_{n-1}) + (1-\lambda) p(w_n|w_2,\ldots,w_{n-1})$$

You can also smooth document estimates with corpus-wide estimates.

$$\hat{p}(w|d) = \lambda \frac{tf_{w,d}}{|d|} + (1-\lambda)\frac{cf_w}{\sum_w cf_w}$$

Relationship to Laplace Smoothing

Most smoothing techniques amount to finding a particular value for λ in Jelinek-Mercer smoothing.

For instance, add-one smoothing is Jelinek-Mercer smoothing with a uniform background distribution and a particular value of λ.

$$\text{Pick } \lambda = \frac{|d|}{|d| + |V|}$$

$$\begin{aligned}
\hat{p}(w|d) &= \lambda \frac{tf_{w,d}}{|d|} + (1-\lambda)\frac{1}{|V|} \\
&= \left(\frac{|d|}{|d| + |V|}\right)\frac{tf_{w,d}}{|d|} + \left(\frac{|V|}{|d| + |V|}\right)\frac{1}{|V|} \\
&= \frac{tf_{w,d}}{|d| + |V|} + \frac{1}{|d| + |V|} \\
&= \frac{tf_{w,d} + 1}{|d| + |V|}
\end{aligned}$$

Relationship to TF-IDF

TF-IDF is also closely related to Jelinek-Mercer smoothing.

If you smooth the query likelihood model with a corpus-wide background probability, the resulting scoring function is proportional to TF and inversely proportional to DF.

$$
\begin{aligned}
\log P(q|d) &= \sum_{w \in q} \log\left(\lambda \frac{tf_{w,d}}{|d|} + (1-\lambda)\frac{df_w}{|c|}\right) \\
&= \sum_{w \in q: tf_{w,d} > 0} \log\left(\lambda \frac{tf_{w,d}}{|d|} + (1-\lambda)\frac{df_w}{|c|}\right) + \sum_{w \in q: tf_{w,d} = 0} \log(1-\lambda)\frac{df_w}{|c|} \\
&= \sum_{w \in q: tf_{w,d} > 0} \log\left(\frac{\lambda \frac{tf_{w,d}}{|d|} + (1-\lambda)\frac{df_w}{|c|}}{(1-\lambda)\frac{df_w}{|c|}}\right) + \sum_{w \in q} \log(1-\lambda)\frac{df_w}{|c|} \\
&\overset{rank}{=} \sum_{w \in q: tf_{w,d} > 0} \log\left(\frac{\lambda \frac{tf_{w,d}}{|d|}}{(1-\lambda)\frac{df_w}{|c|}} + 1\right)
\end{aligned}
$$

Dirichlet Smoothing

Dirichlet Smoothing is the same as Jelinek-Mercer smoothing, picking λ based on document length and a parameter μ – an estimate of the average doc length.

$$\lambda = 1 - \frac{\mu}{|d| + \mu}$$

The scoring function to the right is the Bayesian posterior using a Dirichlet prior with parameters:

$$\left(\mu \frac{cf_{w_1}}{\sum_w cf_w}, \ldots, \mu \frac{cf_{w_n}}{\sum_w cf_w} \right)$$

$$\hat{p}(w|d) = \frac{tf_{w,d} + \mu \frac{cf_w}{\sum_w cf_w}}{|d| + \mu}$$

$$\log p(q|d) = \sum_{w \in q} \log \frac{tf_{w,d} + \mu \frac{cf_w}{\sum_w cf_w}}{|d| + \mu}$$

Example: Dirichlet Smoothing

Query: “president lincoln”

tf	15
cf	160,000
tf	25
cf	2,400
|d|	1,800
Σ	10
μ	2,000

$$\log p(q|d) = \sum_{w \in q} \log \frac{tf_{w,d} + \mu \frac{cf_w}{\sum_w cf_w}}{|d| + \mu}$$

$$= \log \frac{15 + 2,000 \times (160,000/10^9)}{1,800 + 2,000}$$

$$+ \log \frac{25 + 2,000 \times (2,400/10^9)}{1,800 + 2,000}$$

$$= \log(15.32/3,800) + \log(25.005/3,800)$$

$$= -5.51 + -5.02$$

$$= -10.53$$

Effect of Dirichlet Smoothing

Dirichlet Smoothing is a good choice for many IR tasks.

- As with all smoothing techniques, it never assigns zero probability to a term.
- It is a Bayesian posterior which considers how the document differs from the corpus.
- It normalizes by document length, so estimates from short documents and long documents are comparable.
- It runs quickly, compared to many more exotic smoothing techniques.

tf	*tf*	ML Score	Smoothed Score
15	25	-3.937	-10.53
15	1	-5.334	-13.75
15	0	N/A	-19.05
1	25	-5.113	-12.99
0	25	N/A	-14.4

Witten-Bell Smoothing

Dirichlet Smoothing is the same as Jelinek-Mercer smoothing, picking λ based on

* doc length |d|

* doc vocabulary |V| (number of unique terms in document)

$$\lambda = \frac{|d|}{|d| + |V|}$$

N-grams and Skip-grams

An *n-gram* is an ordered set of n contiguous words, usually found within a single sentence. Special cases are $n = 1$ (unigrams), $n = 2$ (bigrams), and $n = 3$ (trigrams).

Skip-grams are more "relaxed" – they can appear in any order, and need not be adjacent. They are an unordered set of n words that appear within a fixed window of k words.

Sentence

The quick brown fox jumped over the lazy dog.

Trigrams (n = 3)

the quick brown
quick brown fox
brown fox jumped
…

Skip-grams (n = 3, k = 5)

quick brown fox
fox jumped quick
lazy dog jumped
…

Markov Chains

We typically construct a generative model of n-grams using *Markov chains* – what is the probability distribution over the next word in the n-gram, given the $n - 1$ words we've seen so far?

$$P(w_n | w_1, w_2, \ldots, w_{n-1})$$

This assumes that words are independent, given the relevance label *and the preceding $n - 1$ words*.

We use a special token, like \$, for words "before" the beginning of the sentence.

Sentence

The quick brown fox jumped over the lazy dog.

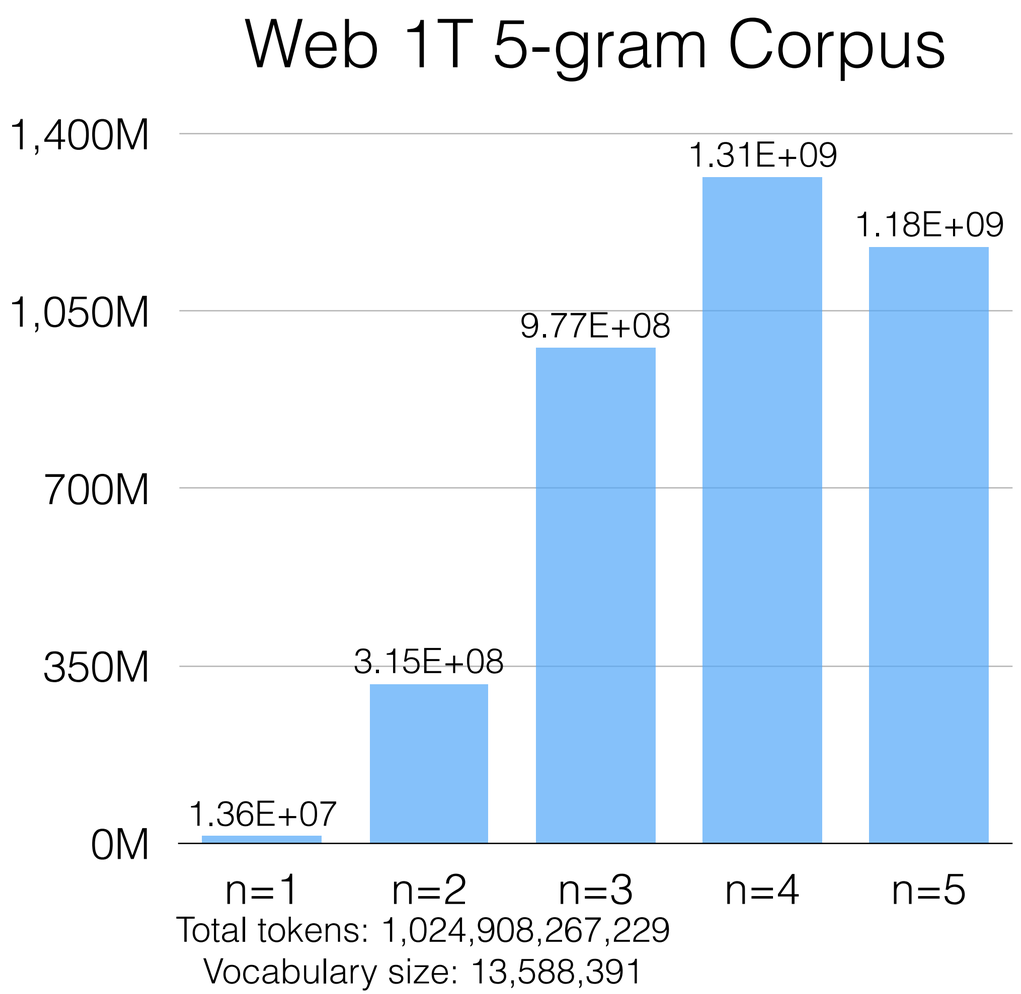
Trigram Sentence Probability

$$P(the|\$, \$) \cdot P(quick|\$, the) \cdot P(brown|the, quick)$$
$$\cdot P(fox|quick, brown) \cdot P(jumped|brown, fox)$$
$$\cdot P(over|fox, jumped) \cdot P(the|jumped, over)$$
$$\cdot P(lazy|over, the) \cdot P(dog|the, lazy)$$

Number of n-grams in a Corpus

How many n-grams do we expect to see, as a function of the vocabulary size v and n-gram size n?

- At first glance, you'd expect to see $\binom{v}{n} = O(v^n)$
- However, most possible n-grams will never appear (like "correct horse battery staple?"), and n-grams are limited by typical sentence lengths.
- As n increases, the number of distinct observed n-grams peaks around $n = 4$ and then decreases.



Choosing n-gram Size

The best n-gram size to use depends on a variance-bias tradeoff:

- Smaller values of n have more training data: infrequent n-grams will appear more often, reducing the *variance* of your probability estimates.
- Larger values of n take more context into account: they have more semantic information, reducing the *bias* of your probability estimates.

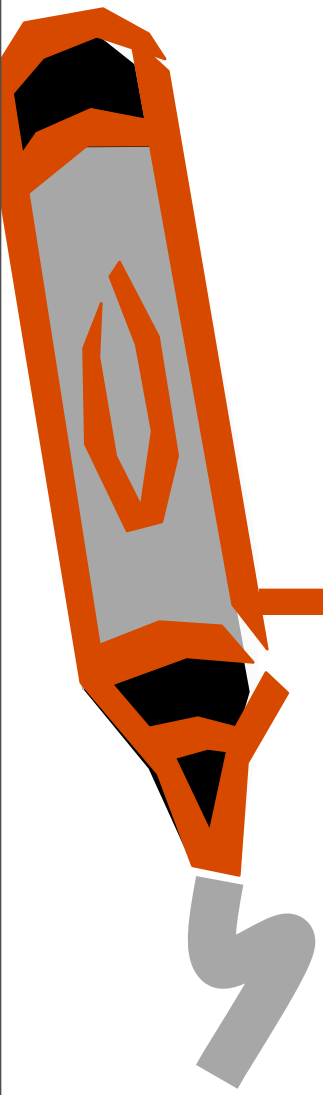
The best n-gram size is the largest value your data will support. Common choices are $n = 3$ for millions of words, or $n = 2$ for smaller corpora.

Wrapping Up

Using n-grams and skip-grams allows us to include some linguistic context in our retrieval models. This helps disambiguate word senses and improve retrieval performance.

Larger values of n are beneficial, if you have the data to support them. The number of n-grams does not grow exponentially in n, so the index size can be manageable.

Next, we'll see how to use an n-gram language model for retrieval.



language models for retrieval

many slides courtesy James Allan@umass Amherst
some slides courtesy Christopher Manning and Prabhakar Raghavan @ Stanford

1

what is a retrieval model?

- Model is an idealization or abstraction of an actual process
- Mathematical models are used to study the properties of the process, draw conclusions, make predictions
- Conclusions derived from a model depend on whether the model is a good approximation of the actual situation
- Statistical models represent repetitive processes, make predictions about frequencies of interesting events
- Retrieval models can describe the computational process
  - e.g. how documents are ranked
  - Note that how documents or indexes are *stored* is implementation
- Retrieval models can attempt to describe the human process
  - e.g. the information need, interaction
  - Few do so meaningfully
- Retrieval models have an explicit or implicit definition of relevance

2

retrieval models

- boolean
- vector space
- latent semnatic indexing

today

- statistical language
- inference network
- hyperlink based

3

language models not in MIR

1. J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. Proceedings of ACM-SIGIR 1998, pages 275-281.
2. J. M. Ponte. A language modeling approach to information retrieval. Phd dissertation, University of Massachusets, Amherst, MA, September 1998.
3. D. Hiemstra. Using Language Models for Information Retrieval. PhD dissertation, University of Twente, Enschede, The Netherlands, January 2001.
4. D. R. H. Miller, T. Leek, and R. M. Schwartz. A hidden Markov model information retrieval system. Proceedings of ACM-SIGIR 1999, pages 214-221.
5. F. Song and W. B. Croft. A general language model for information retrieval. In Proceedings of Eighth International Conference on Information and Knowledge Management (CIKM 1999)
6. S. F. Chen and J. T. Goodman. An empirical study of smoothing techniques for language modeling. In Proceedings of the 34th Annual Meeting of the ACL, 1996.
7. C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. Proceedings of the ACM-SIGIR 2001, pages 334-342.
8. V. Lavrenko and W. B. Croft. Relevance-based language models. Proceedings of the ACM SIGIR 2001, pages 120-127.
9. V. Lavrenko and W. B. Croft, Relevance Models in Information Retrieval, in Language Modeling for Information Retrieval, W. Bruce Croft and John Lafferty, ed., Kluwer Academic Publishers, chapter 2.

4

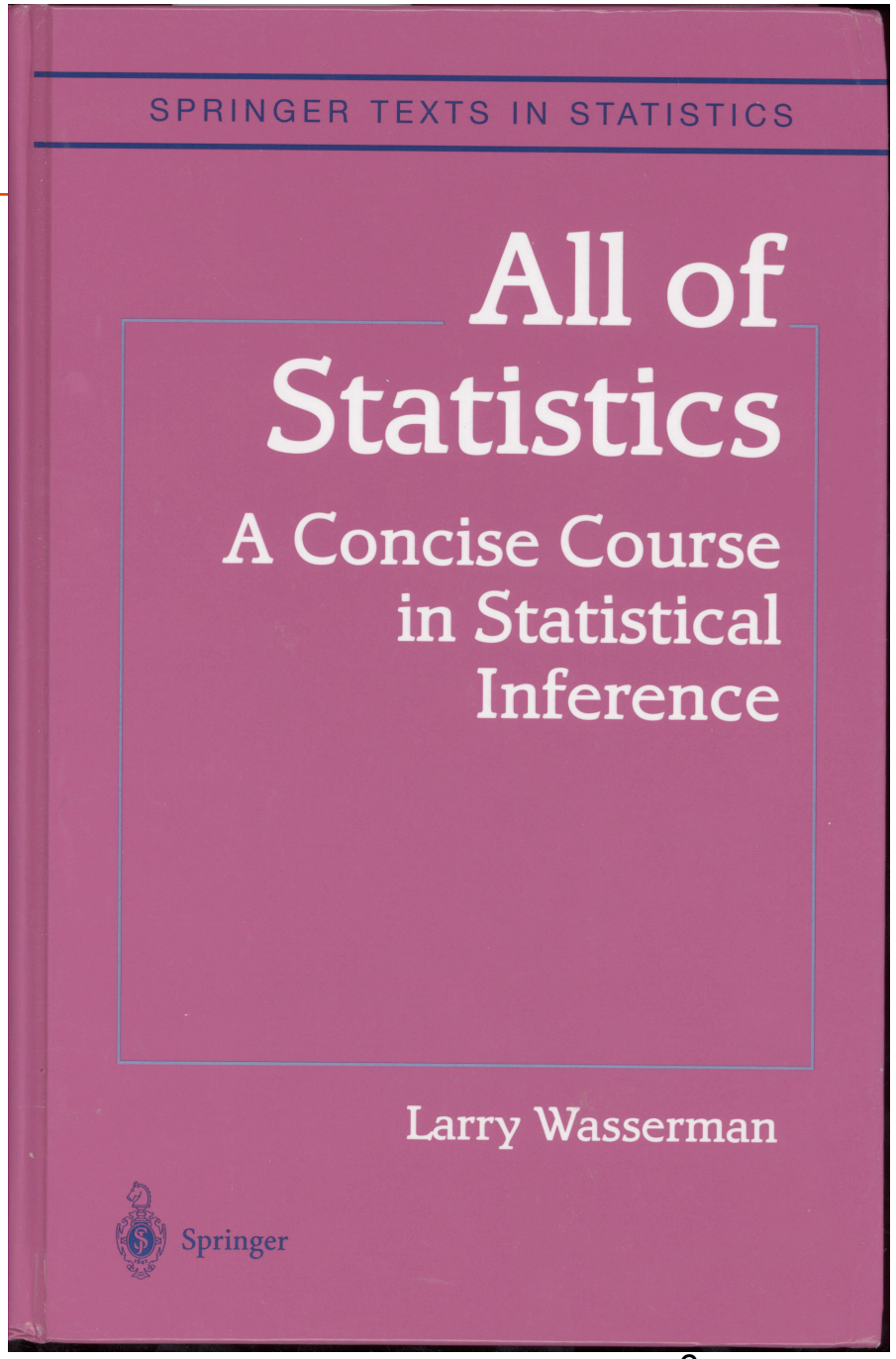
outline

- review: probabilities
- language model
- similarity, ranking in LM
- probability estimation
- smoothing methods
- examples

5

probabilities

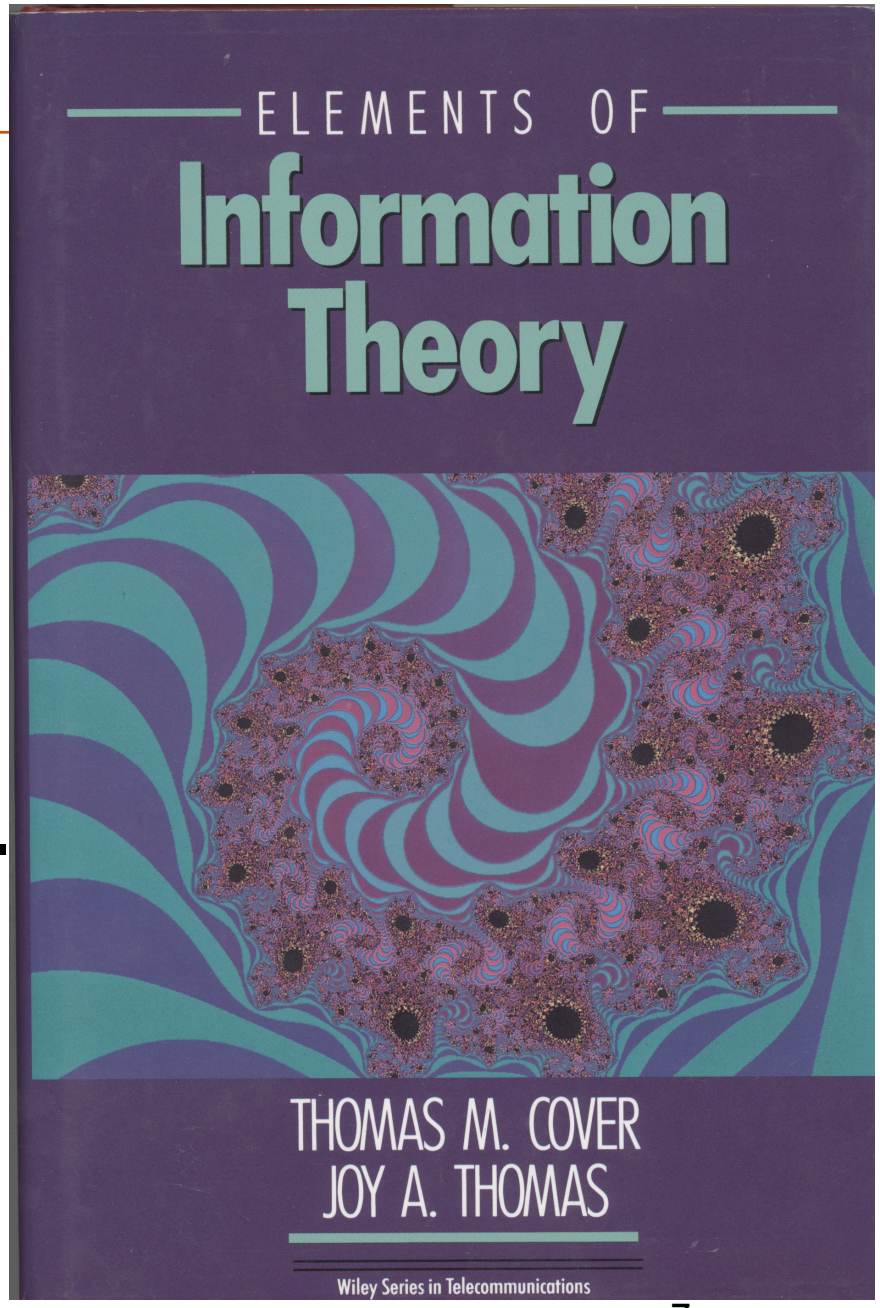
- sample space
- probability
- independent events
- cond. probability
- Bayes theorem
- distributions



6

information theory, coding

- entropy
- joint entropy
- cond. entropy
- relative entropy
- convexity, Jensen ineq.
- optimal coding
- Fano's ineq.



7

outline

- review: probabilities
- language model
- similarity, ranking in LM
- probability estimation
- smoothing methods
- examples

8

what is a language model ?

- Probability distribution over strings of text
  - how likely is a given string (observation) in a given “language”
  - for example, consider probability for the following four strings

p1 = P(“a quick brown dog”)
p2 = P(“dog quick a brown”)
p3 = P(“быстрая brown dog”)
p4 = P(“быстрая собака”)

  - English: p1 > p2 > p3 > p4

- ... depends on what “language” we are modeling
  - In most of IR, assume that p1 == p2

9

lang modeling for IR

- Every document in a collection defines a “language”
  - consider all possible sentences (strings) that author could have written down when creating some given document
  - some are perhaps more likely to occur than others
- subject to topic, writing style, language ...
  - P(s|MD) = probability that author would write down string “s”
- think of writing a billion variations of a document and counting how many time we get “s”
- Now suppose “Q” is the user’s query
  - what is the probability that author would write down “q” ?
- Rank documents D in the collection by P(Q|MD)
  - probability of observing “Q” during random sampling from the language model of document D

10

language models

- estimate probabilities of certain "events" in the text

- based on these probabilities, use likelihood as similarity

- language model based on
  - letters?
  - words?
  - phrases?

"generator"

generative model

task: write an automatic text generator

=> probabilistic generator

which unit for probs?

11

statistical text generation

1. *Zero-order approximation.* (The symbols are independent and equiprobable.)

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ

   FFJEYVKCQSGXYD QPAAMKBZAACIBZLHJQD

2. *First-order approximation.* (The symbols are independent. Frequency of letters matches English text.)

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI

   ALHENHTTPA OOBTTVA NAH BRL

3. *Second-order approximation.* (The frequency of pairs of letters matches English text.)

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY

   ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO

   TIZIN ANDY TOBE SEACE CTISBE

4. *Third-order approximation.* (The frequency of triplets of letters matches English text.)

   IN NO IST LAT WHEY CRATICT FROURE BERS GROCID

   PONDENOME OF DEMONSTURES OF THE REPTAGIN IS

   REGOACTIONA OF CRE

12

what prob model?

- letters?
- words? unigrams
- phrases

5. *Fourth-order approximation.* (The frequency of quadruplets of letters matches English text. Each letter depends on the previous three letters. This sentence is from Lucky's book, *Silicon Dreams* [183].) P(D|"XYZT") = correct

THE GENERATED JOB PROVIDUAL BETTER TRAND THE DISPLAYED CODE, ABOVERY UPONDULTS WELL THE CODERST IN THESTICAL IT DO HOCK BOTHE MERG. (INSTATES CONS ERATION. NEVER ANY OF PUBLE AND TO THEORY. EVENTIAL CALLEGAND TO ELAST BENERATED IN WITH PIES AS IS WITH THE)

big jump

Instead of continuing with the letter models, we jump to word models.

6. *First-order word model.* (The words are chosen independently but with frequencies as in English.) P(words) = correct

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

7. *Second-order word model.* (The word transition probabilities match English text.) P("XYZ ABC") = correct

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

8. word-trigrams 9. phrases

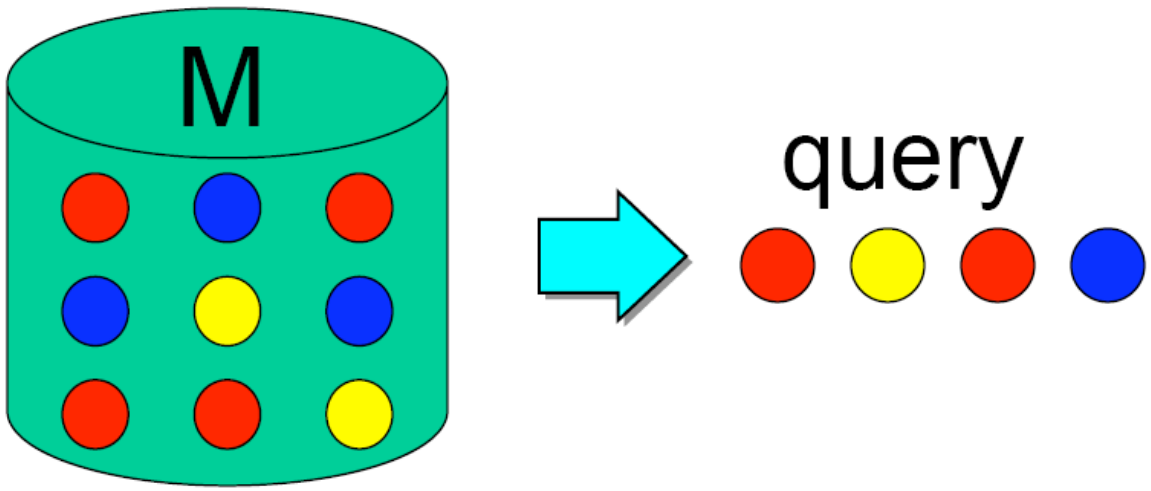
LM choices

- What kind of language model should we use?
  - Unigram or higher-order models?
  - Multinomial or multiple-Bernoulli?
- How can we estimate model parameters?
  - Basic models
  - Translation models
  - Aspect models
  - non-parametric models
- How can we use the model for ranking?
  - Query-likelihood
  - Document-likelihood
  - Likelihood Ratio
  - Divergence of query and document models

14

unigram LM

- words are sampled independently, with replacement
- order of the words is lost (no phrases)



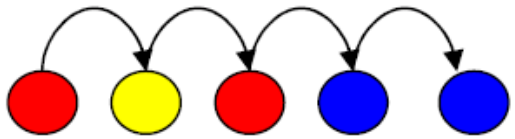
$$P(\bullet\bullet\bullet\bullet) = P(\bullet)P(\bullet)P(\bullet)P(\bullet)$$

$$= 4/9 * 2/9 * 4/9 * 3/9$$

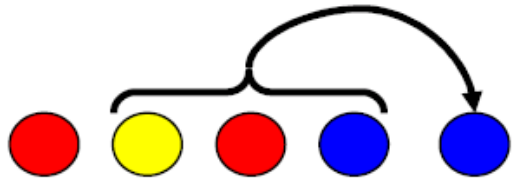
15

higher-order LM

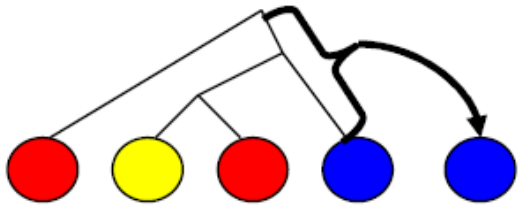
- Unigram model assumes word independence
  - cannot capture surface form: P("brown dog") == P("dog brown")
- Higher-order models
  - n-gram: condition on preceding words



  - cache: condition on a window (cache)



  - grammar: condition on parse tree



- Are they useful?
  - no improvements from n-gram, grammar-based models
  - some research on cache-like models (proximity, passages, etc.)
  - parameter estimation is prohibitively expensive

16

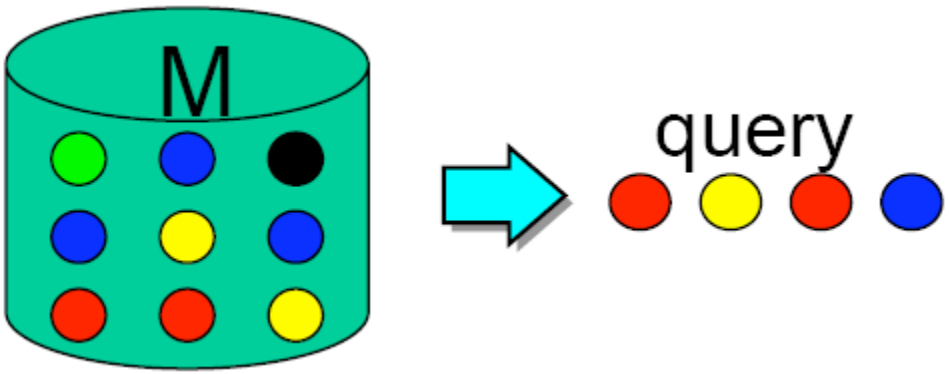
outline

- review: probabilities
- language model
- similarity, ranking in LM
- probability estimation
- smoothing methods
- examples

17

multinomial similarity

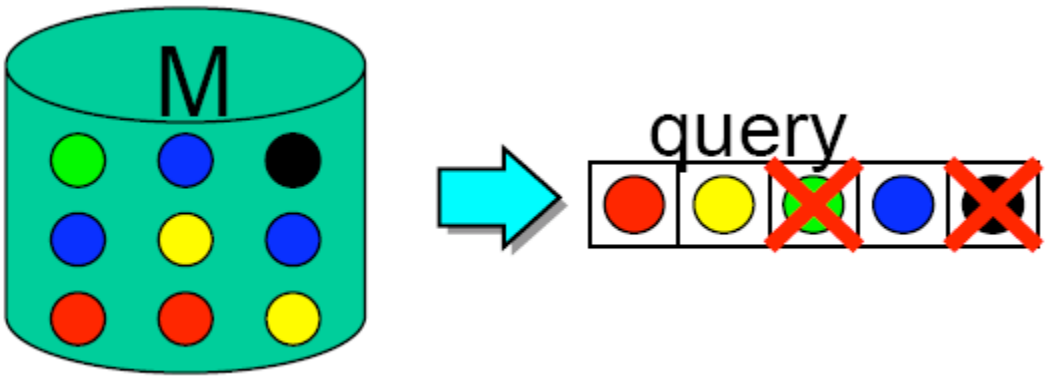
- Predominant model
- Fundamental event:
what is the identity of the i'th query token?
- observation is a sequence of events, one for each query token



$$P(q_1 \ldots q_k \mid M) = \prod_{i=1}^{k} P(q_i \mid M)$$

multiple-Bernoulli similarity

- Original model
- fundamental event: *does the word w occur in the query?*
- observation is a vector of binary events, one for each possible word



$$P(q_1 \ldots q_k \mid M) = \prod_{w \in q_1 \ldots q_k} P(w \mid M) \prod_{w \notin q_1 \ldots q_k} [1 - P(w \mid M)]$$

19

score, ranking in LM

- what is the probability to generate the given query, given a language model?
- what is the probability to generate the given document, given a language model?
- how ”close” are 2 statistical models?

20

score: query likelihood

- Standard approach: query-likelihood
  - estimate a language model MD for every document D in the collection
  - rank docs by the probability of “generating” the query

$$P(q_1 \ldots q_k \mid M_D) = \prod_{i=1}^{k} P(q_i \mid M_D)$$

- |
  - no notion of relevance in the model: everything is random sampling
  - user feedback / query expansion not part of the model
  -examples of relevant documents cannot help us improve the language model MD
  - does not directly allow weighted or structured queries

21

score: document likelihood

- Flip the direction of the query-likelihood approach
  - estimate a language model MQ for the query Q
  - rank docs D by the likelihood of being a random sample from MQ
  - MQ expected to "predict" a typical relevant document

- Problems:
  - different doc lengths, probabilities not comparable
  - favors documents that contain frequent (low content) words

$$P(D \mid M_Q) = \prod_{w \in D} P(w \mid M_Q)$$

22

score: likelihood ratio

- Try to fix document likelihood:
  - Bayes' likelihood that M_q was the source, given that we observed D
  - related to Probability Ranking Principle: P(D|R) / P (D|N)
  - allows relevance feedback, query expansion, etc.
  - can benefit from complex estimation of the query model MQ

$$P(M_Q \mid D) = \frac{P(M_Q)P(D \mid M_Q)}{P(D)} \approx \frac{c\prod_{w \in D} P(w \mid M_Q)}{\prod_{w \in D} P(w \mid GE)}$$

23

score: model comparison

- Combine advantages of two ranking methods
  - estimate a model of both the query MQ and the document MD
  - directly compare similarity of the two models
  - natural measure of similarity is cross-entropy (others exist):

$$H(M_Q \| M_D) = -\sum_w P(w|M_Q) \log P(w|M_D)$$

  - number of bits we would need to "encode" MQ using MD
  - equivalent to Kullback-Leibler divergence
  - equivalent to query-likelihood if MQ is simply counts of words in Q

- Cross-entropy is not symmetric: use H (MQ || MD)
  - reverse works consistently worse, favors different document
  - use reverse if ranking multiple queries w.r.t. one document

Q.L. model

score(d) ≈ $\sum_{w \in q}$ log [P(w|d)]

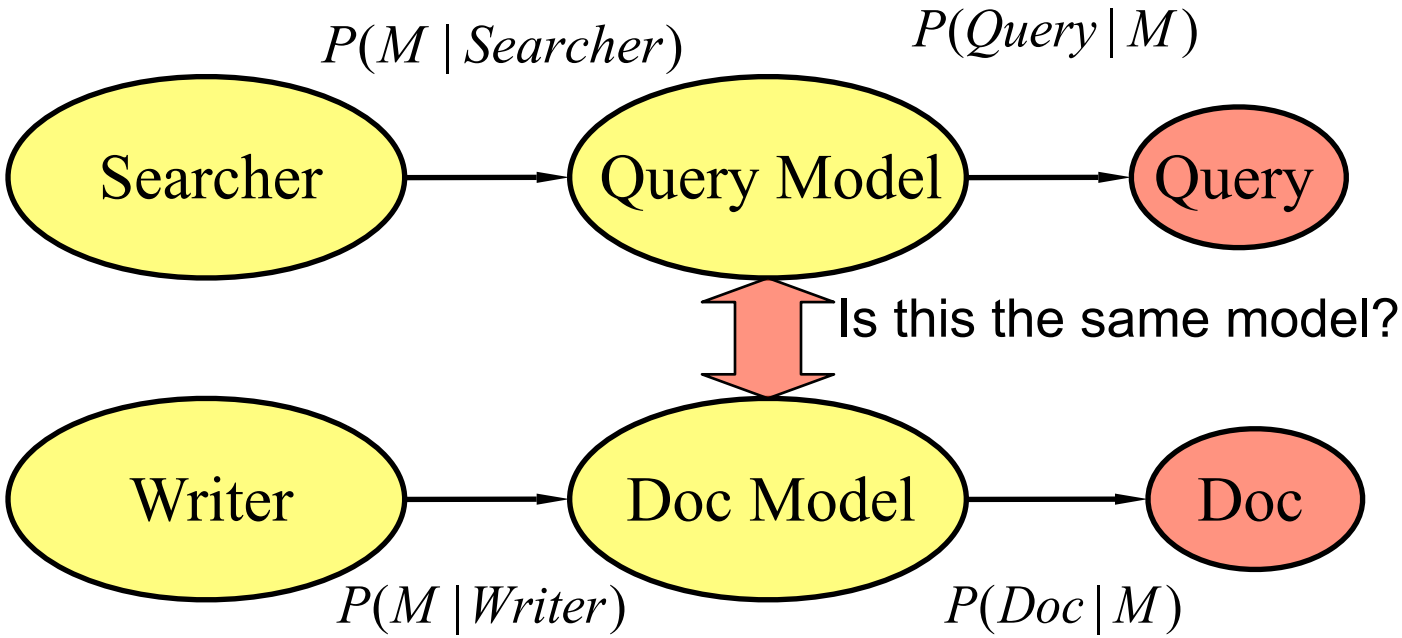
basic estim

w ∉ d ⇒ P(w|d) = 0 ⇒ log(P(w|d)) = −∞

$\frac{TF(w,d)}{DL(d)}$

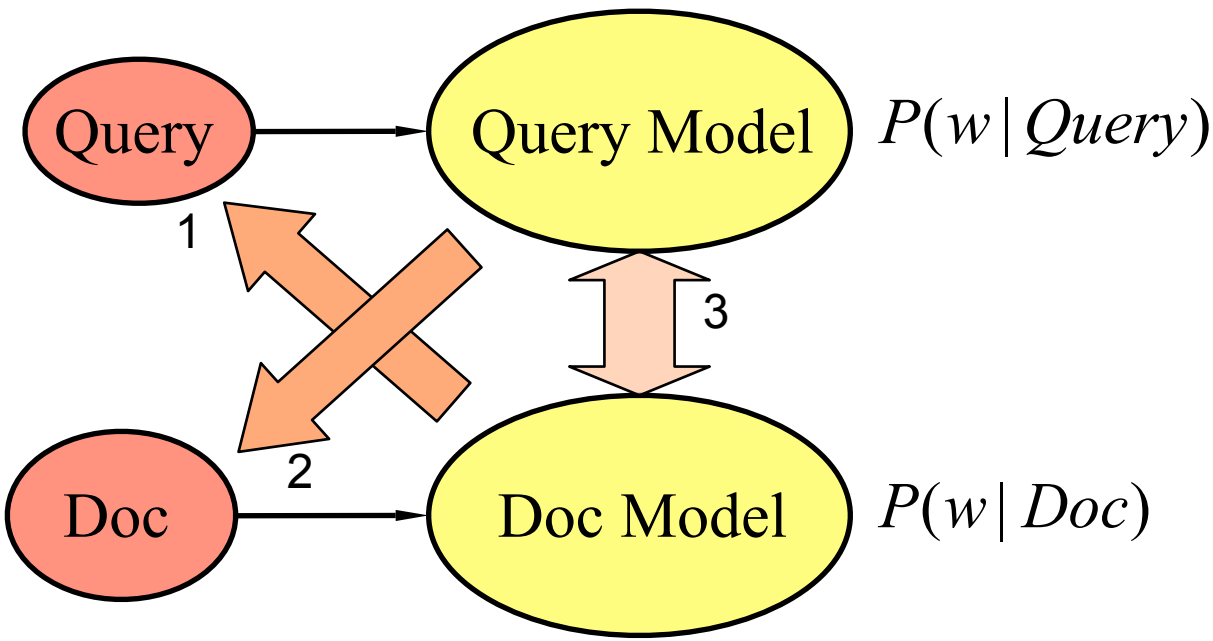
24

Models of Text Generation



25

Retrieval with Language Models



Retrieval: Query likelihood (1)
Document likelihood (2)
Model comparison (3)

26

LM: popular choices

- Use Unigram models
  - no consistent benefit from using higher order models
  - estimation is much more complex (e.g. bi-gram from a 3-word query)
- Use Multinomial models
  - well-studied, consistent with other fields that use LMs
  - extend multiple-Bernoulli model to non-binary events?
- Use Model Comparison for ranking
  - allows feedback, expansion, etc. through estimation of MQ and MD
  - use KL(MQ || MD) for ranking multiple documents against a query
- Estimation of MQ and MD is a crucial step
  - very significant impact on performance (more than other choices)
  - key to cross-language, cross-media and other applications

27

Translation model (Berger and Lafferty)

- Basic LMs do not address issues of synonymy.
  - Or any deviation in expression of information need from language of documents
- A translation model lets you generate query words not in document via "translation" to synonyms etc.
  - Or to do cross-language IR, or multimedia IR

$$P(\vec{q} \mid M) = \prod_i \sum_{v \in Lexicon} P(v \mid M) T(q_i \mid v)$$

Basic LM Translation

  - Need to learn a translation model (using a dictionary or via statistical machine translation)

28

outline

- review: probabilities
- language model
- similarity, ranking in LM
- probability estimation
- smoothing methods
- examples

29

N=3
n=2 ⇒ p(wad) = 2/3

same problem for very short docs

estimation

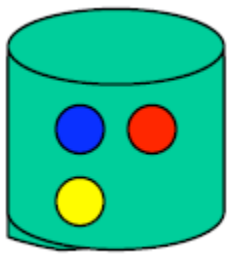
- Want to estimate MQ and/or MD from Q and/or D
- General problem:
  - given a string of text S (= Q or D), estimate its language model MS
  - S is commonly assumed to be an i.i.d. random sample from MS
    - Independent and identically distributed
- Basic Language Models
  - maximum-likelihood estimator and the zero frequency problem
  - discounting, interpolation techniques
  - Bayesian estimation

30

maximum likelihood

- count relative frequencies of words in S
- maximum-likelihood property:
  - assigns highest possible likelihood to the observation
- unbiased estimator:
  - if we repeat estimation an infinite number of times with different starting points S, we will get correct probabilities (on average)
  - this is not very useful…

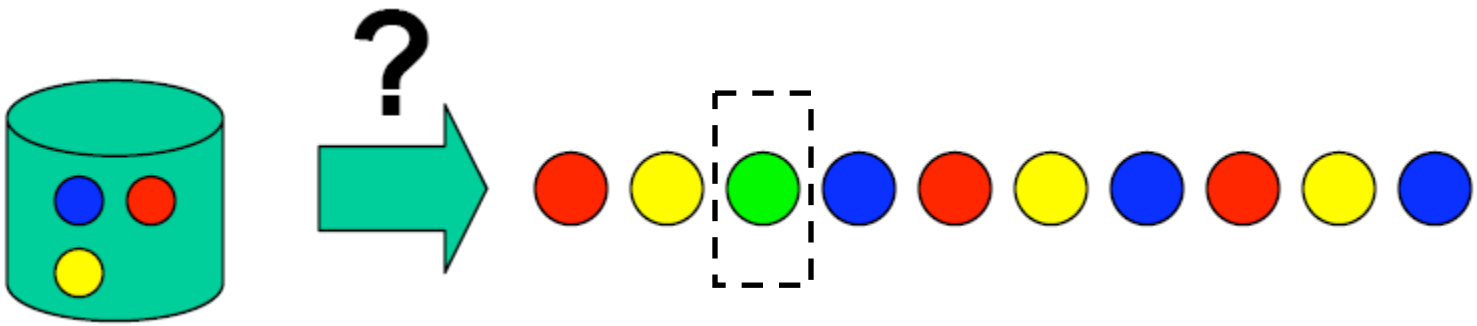
$$P_{ml}(w|M_S) = \#(w,S) / |S|$$



P (●) = 1/3
P (●) = 1/3
P (●) = 1/3
P (●) = 0
P (●) = 0

zero-frequency problem

- Suppose some event not in our observation S
  - Model will assign zero probability to that event
  - And to any set of events involving the unseen event
- Happens very frequently with language
- It is incorrect to infer zero probabilities
  - especially when creating a model from short samples



32

Discount-smooth

Laplace smoothing

derived very complicated (with integrals)

- count events in observed data
- add 1 to every count
- renormalize to obtain probabilities
- it corresponds to uniform priors

K = #possible = 2

coins Laplace

N trial H+T=N

heads tails

Laplace estimate

$$\left[\frac{H+1}{N+2}, \frac{TH}{N+2}\right]$$

N trials

counts

- if event counts are $(m_1, m_2, \ldots, m_k)$ with $\sum_i m_i = N$ then

max lielihood estimates are $(\frac{m_1}{N}, \frac{m_2}{N}, \ldots, \frac{m_k}{N})$

laplace estimates are $(\frac{m_1+1}{N+k}, \frac{m_2+1}{N+k}, \ldots, \frac{m_k+1}{N+k})$

basic estimates = Max likelihood

Laplace estimates

$$\frac{m_1}{N} \quad \frac{m_2}{N} \quad \cdots \quad \frac{m_k}{N} \Longrightarrow \frac{m_1+1}{N+k} \quad \frac{m_2+1}{N+k} \quad \cdots \quad \frac{m_k+1}{N+k}$$

+1

+k

$\sum_i \frac{m_i}{N} = 1$? yes $\sum(\,) = \frac{\sum m_i}{N} = \frac{N}{N} = 1$

33

Laplace dtg vol $N = 1000$ $u_1 = \#1$ $u_2 = \#2$, .. $u_6 = \#6$

$k = 6$

Direct/Basi/ Max-Likeli/ $\left(\frac{u_1}{N} \quad \frac{u_2}{N}\right) - - - \frac{u_6}{N}$

Laplace $\left(\frac{u_1+1}{N+6} \right) \frac{u_2+1}{N+6}) \cdot - - \frac{u_6+1}{N+6}$

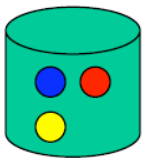
discounting methods

- Laplace smoothing

$$\sum_i \frac{m_i+1}{N+k} = \frac{\sum m_i + \sum 1}{N+k} = \frac{N+k}{N+k} = 1$$

$\Rightarrow$ valid distribution

- Lindstone correction
  - add ε to all count, renormalize



+ ε

P (●) = (1 + ε) / (3+5ε)
P (●) = (1 + ε) / (3+5ε)
P (●) = (1 + ε) / (3+5ε)
P (●) = (0 + ε) / (3+5ε)
P (●) = (0 + ε) / (3+5ε)

- absolute discounting
  - substract ε , redistribute probab mass

$\frac{m_1}{N}$ $\frac{m_2}{N}$ ... $\frac{m_k}{N}$

basic

$\frac{m_1+\varepsilon}{N+k\varepsilon}$ $\frac{m_2+\varepsilon}{N+k\varepsilon}$ ... $\frac{m_k+\varepsilon}{N+k\varepsilon}$

$\frac{1}{k}$ = uniform general Laplace (ε) $\sum 1 = 1$

34

discounting methods

- Held-out estimation
  - Divide data into training and held-out sections
  - In training data, count Nr, the number of words occurring r times
  - In held-out data, count Tr, the number of times those words occur
  - r* = Tr/Nr is adjusted count (equals r if training matches held-out)
  - Use r*/N as estimate for words that occur r times
- Deleted estimation (cross-validation)
  - Same idea, but break data into K sections
  - Use each in turn as held-out data, to calculate Tr(k) and Nr(k)
  - Estimate for words that occur r times is average of each
- Good-Turing estimation
  - From previous, P(w|M) = r* / N if word w occurs r times in sample
  - In Good-Turing, steal total probability mass from next most frequent word
  - Provides probability mass for words that occur r=0 times
  - Take what's leftover from r>0 to ensure adds to one

$$TPM(r+1) = N_{r+1} \cdot \frac{r+1}{N}$$

$$\begin{aligned} P(w_r|M) &= TPM(r+1)/N_r \\ &= \frac{N_{r+1}}{N_r} \cdot \frac{r+1}{N} \end{aligned}$$

35

interpolation methods

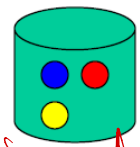
≠ discounting

- Problem with all discounting methods:
  - discounting treats unseen words equally (add or subtract ϵ)
  - some words are more frequent than others
- Idea: use background probabilities
  - "interpolate" ML estimates with General English expectations (computed as relative frequency of a word in a large collection)
  - reflects expected frequency of events

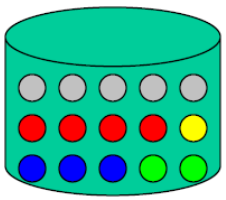
direct

prior

ML estimate

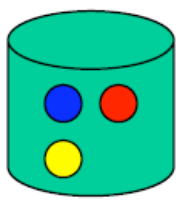
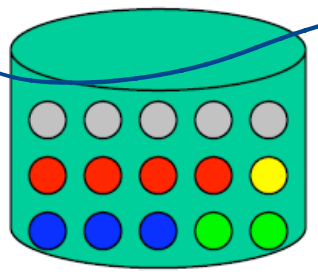


background probability



direct estimate = basic estimate (from doc)

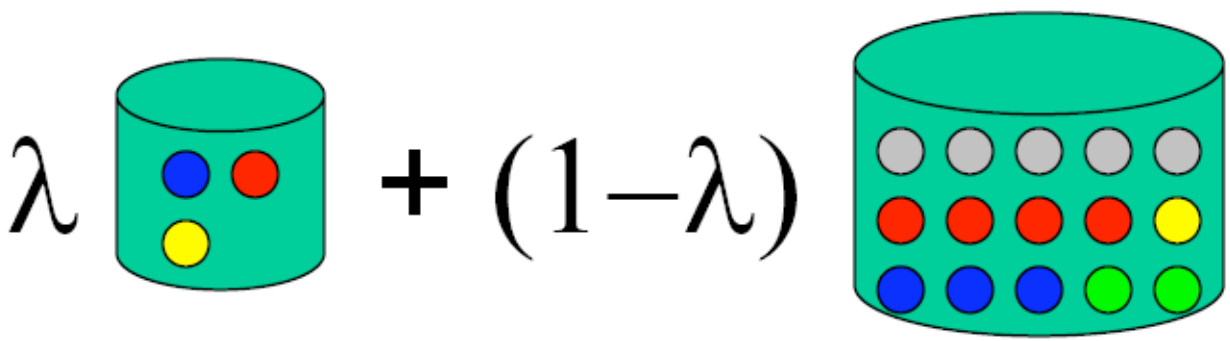
estimate from large data (from collection) (english prior)

final estimate = λ  + $(1-\lambda)$ 

36

Jelinek Mercer smoothing

- Correctly setting **λ** is very important
- Start simple
  - set **λ** to be a constant, independent of document, query
- Tune to optimize retrieval performance
  - optimal value of **λ** varies with different databases, query sets, etc.

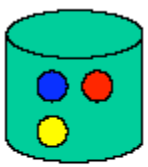


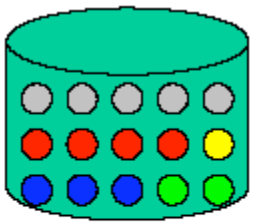
37

Dirichlet smoothing

- Problem with Jelinek-Mercer:
  - longer documents provide better estimates
  - could get by with less smoothing
- Make smoothing depend on sample size
- N is length of sample = document length
- μ is a constant

$$\underbrace{N / (N + \mu)}_{\lambda} \; P(w|d) + \underbrace{\mu / (N + \mu)}_{(1-\lambda)} \; P(w|C)$$



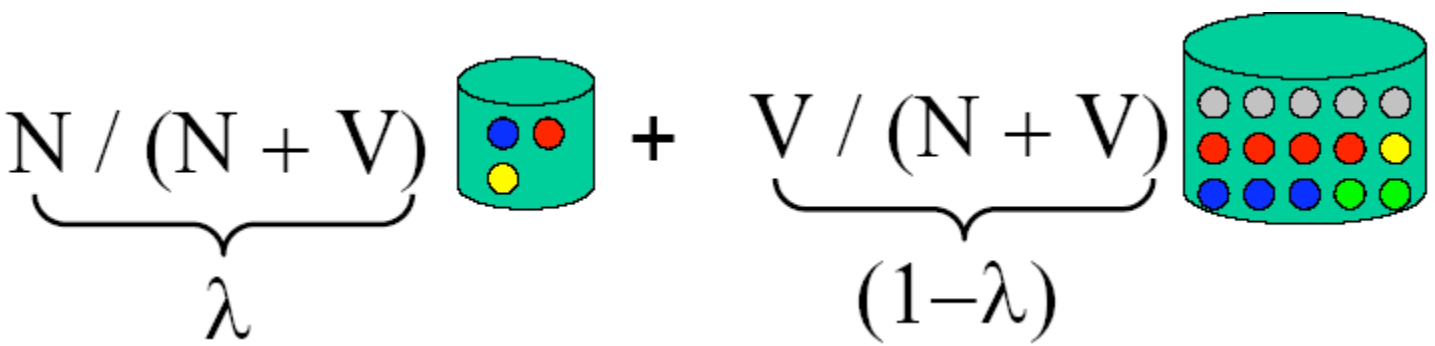


38

Witten-Bell smoothing

- A step further:
  - condition smoothing on "redundancy" of the example
  - long, redundant example requires little smoothing
  - short, sparse example requires a lot of smoothing

- Derived by considering the proportion of new events as we walk through example
  - N is total number of events = document length
  - V is number of unique events = number of unique terms in doc



39

interpolation vs back-off

- Two possible approaches to smoothing
- Interpolation:
  - Adjust probabilities for all events, both seen and unseen
- Back-off:
  - Adjust probabilities only for unseen events
  - Leave non-zero probabilities as they are
  - Rescale everything to sum to one: rescales "seen" probabilities by a constant
- Interpolation tends to work better
  - And has a cleaner probabilistic interpretation

40

Two-stage smoothing

Query =	"the	algorithms	for	data	mining"
d1:	**0.04**	**0.001**	**0.02**	**0.002**	**0.003**
d2:	**0.02**	**0.001**	**0.01**	**0.003**	**0.004**

41

Two-stage smoothing

Query = "the	algorithms	for	data	mining"
d1:	**0.04**	**0.001**	**0.02**	**0.002**	**0.003**
d2:	**0.02**	**0.001**	**0.01**	**0.003**	**0.004**

p("algorithms"|d1) = p("algorithm"|d2)

p("data"|d1) < p("data"|d2)

p("mining"|d1) < p("mining"|d2)

But p(q|d1)>p(q|d2)!

41

Two-stage smoothing

Query = "the	algorithms	for	data	mining"
d1:	**0.04**	**0.001**	**0.02**	**0.002**	**0.003**
d2:	**0.02**	**0.001**	**0.01**	**0.003**	**0.004**

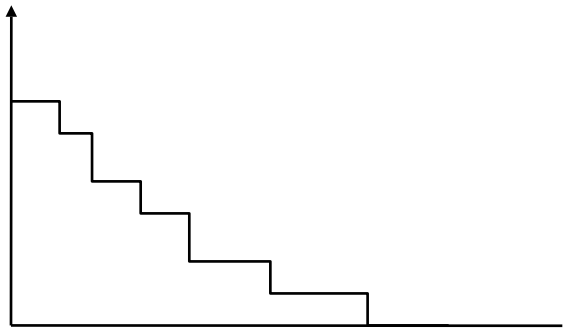
p("algorithms"|d1) = p("algorithm"|d2)
p("data"|d1) < p("data"|d2)
p("mining"|d1) < p("mining"|d2)

But p(q|d1)>p(q|d2)!

We should make p("the") and p("for") less different for all docs.

41

Two-stage smoothing



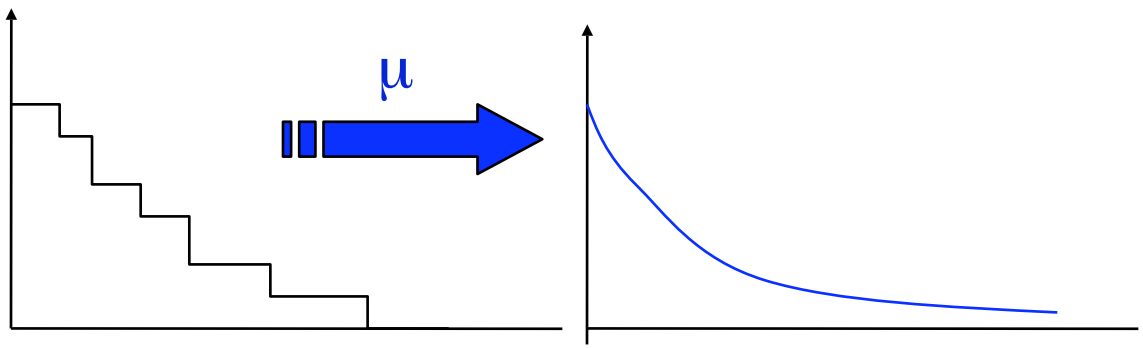
$$P(w|d) = \frac{c(w,d)}{|d|}$$

42

Two-stage smoothing

Stage-1

-Explain unseen words
-Dirichlet prior(Bayesian)



$$P(w|d) = \frac{c(w,d) + \mu p(w|C)}{|d| + \mu}$$

42

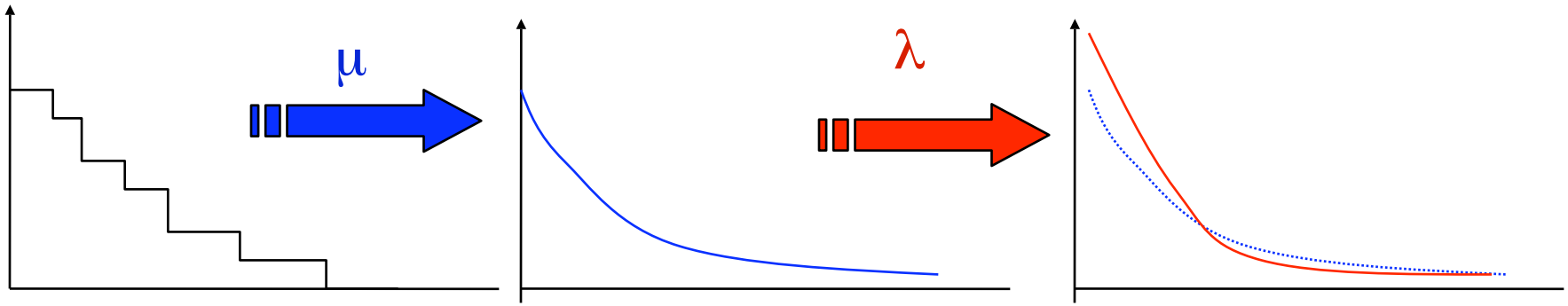
Two-stage smoothing

Stage-1

-Explain unseen words
-Dirichlet prior(Bayesian)

Stage-2

-Explain noise in query
-2-component mixture



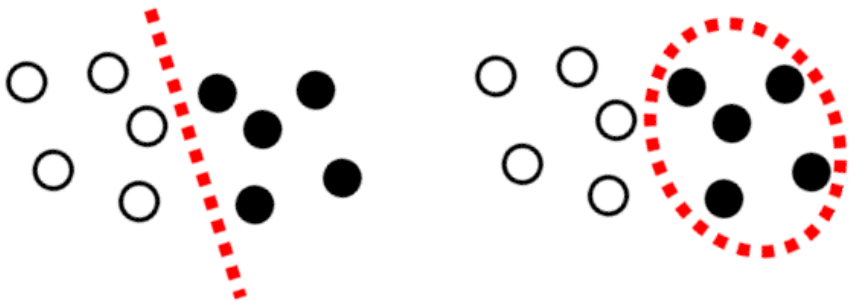
$$P(w|d) = (1-\lambda)\frac{c(w,d)+\mu p(w|C)}{|d|+\mu} + \lambda p(w|U)$$

42

LM are generative techniques

- How do we determine if a given model is a LM?
- LM is generative
  - at some level, a language model can be used to generate text
  - explicitly computes probability of observing a string of text
  - Ex: probability of observing a query string from a document model
    probability of observing an answer from a question model
  - model an entire population

- Discriminative approaches
  - model just the decision boundary
  - Ex: is this document relevant?
    does it belong to class X or Y



  - have a lot of advantages,
  - but these are not generative approaches

43

LM: summary

- Goal: estimate a model M from a sample text S
- Use maximum-likelihood estimator
  - count the number of times each word occurs in S, divide by length
- Smoothing to avoid zero frequencies
  - discounting methods: add or subtract a constant, redistribute mass
  - better: interpolate with background probability of a word
  - smoothing has a role similar to IDF in classical models
- Smoothing parameters very important
  - Dirichlet works well for short queries (need to tune the parameter)
  - Jelinek-Mercer works well for longer queries (also needs tuning)
  - Lots of other ideas being worked on

44

Language models: pro & con

- Novel way of looking at the problem of text retrieval based on probabilistic language modeling
  - Conceptually simple and explanatory
  - Formal mathematical model
  - Natural use of collection statistics, not heuristics (almost···)
- LMs provide effective retrieval and can be improved to the extent that the following conditions can be met
  - Our language models are accurate representations of the data.
  - Users have some sense of term distribution.

45

Comparison With Vector Space

- There's some relation to traditional tf.idf models:
  - (unscaled) term frequency is directly in model
  - the probabilities do length normalization of term frequencies
  - the effect of doing a mixture with overall collection frequencies is a little like idf: terms rare in the general collection but common in some documents will have a greater influence on the ranking

46

Comparison With Vector Space

- Similar in some ways
  - Term weights based on frequency
  - Terms often used as if they were independent
  - Inverse document/collection frequency used
  - Some form of length normalization useful

- Different in others
  - Based on probability rather than similarity
    - Intuitions are probabilistic rather than geometric
  - Details of use of document length and term, document, and collection frequency differ

47