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

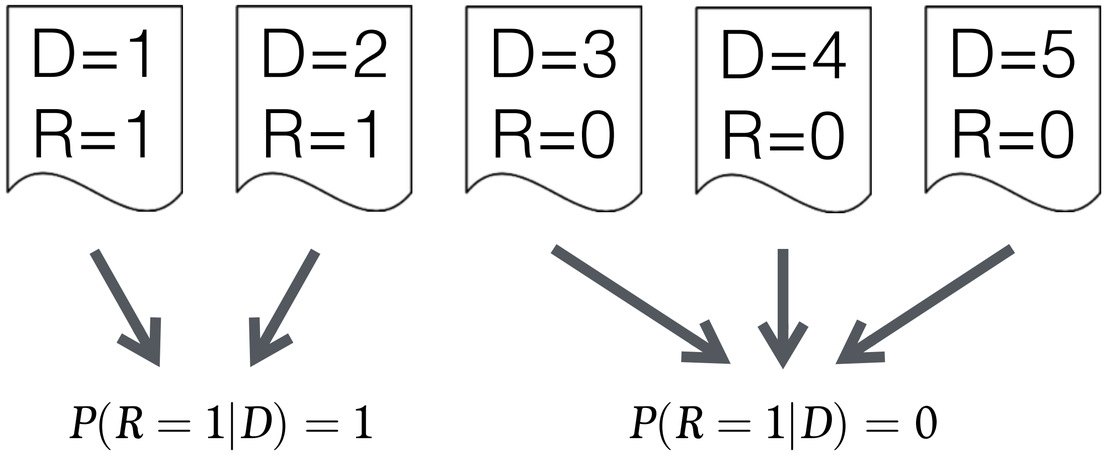
For simplicity here, we will hold the query constant and consider $P(R=1|D)$.

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

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:

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

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

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

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.

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.

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.

$$
\begin{aligned}
P(D|Q) &\overset{rank}{=} P(Q|D)P(D) \\
&= P(Q|D) && \text{Assuming uniform prior} \\
&= \prod_{w \in Q} P(w|D) && \text{Naive Bayes unigram model} \\
&\overset{rank}{=} \sum_{w \in Q} \log P(w|D) && \text{Numerically stable version}
\end{aligned}
$$

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.

Comparing Distributions

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

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

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

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.

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

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

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

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

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

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

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

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(w|q)	p(w|d)	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).

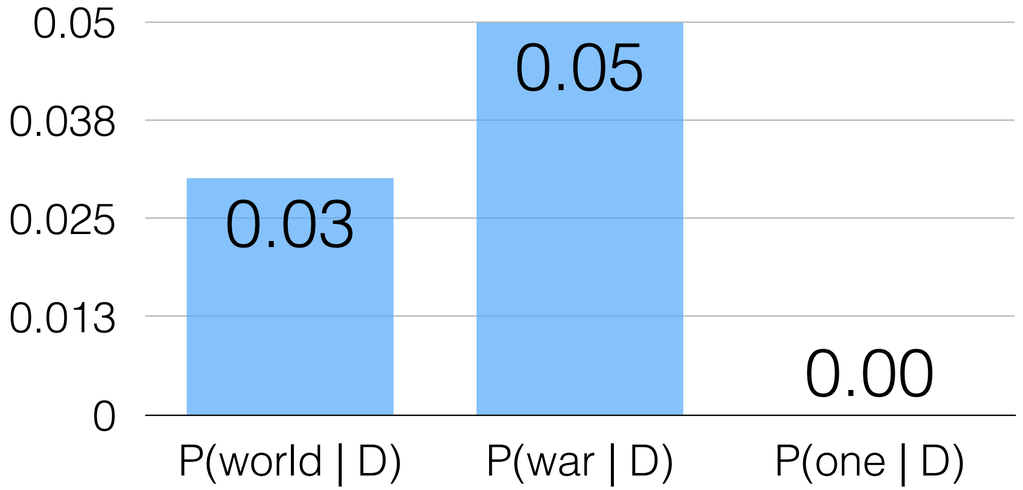
Probability Estimation

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"



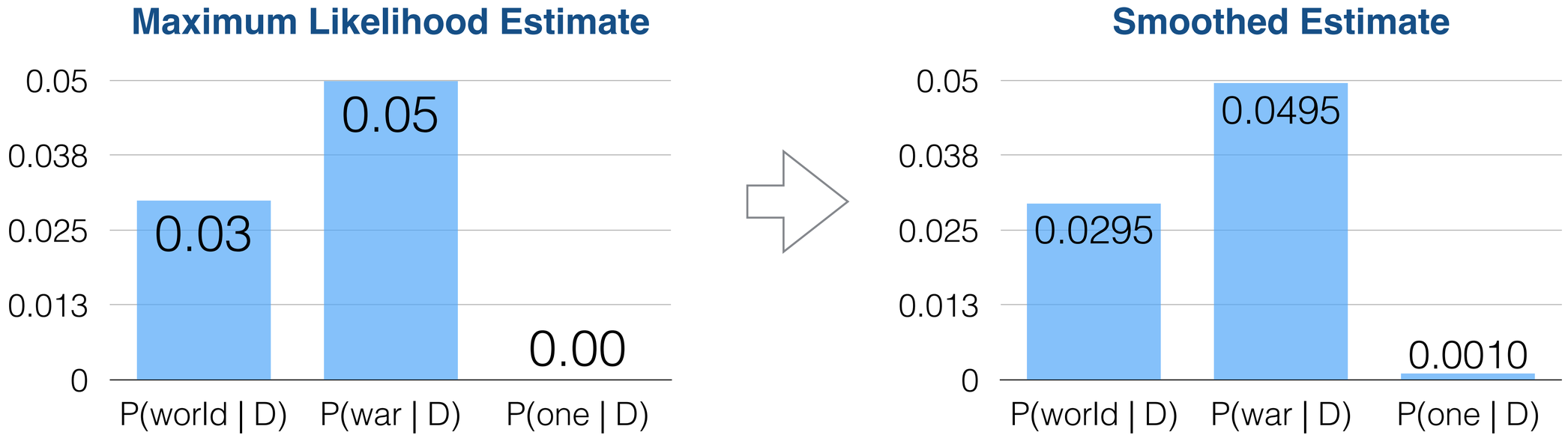
Query Likelihood Model

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

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

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.

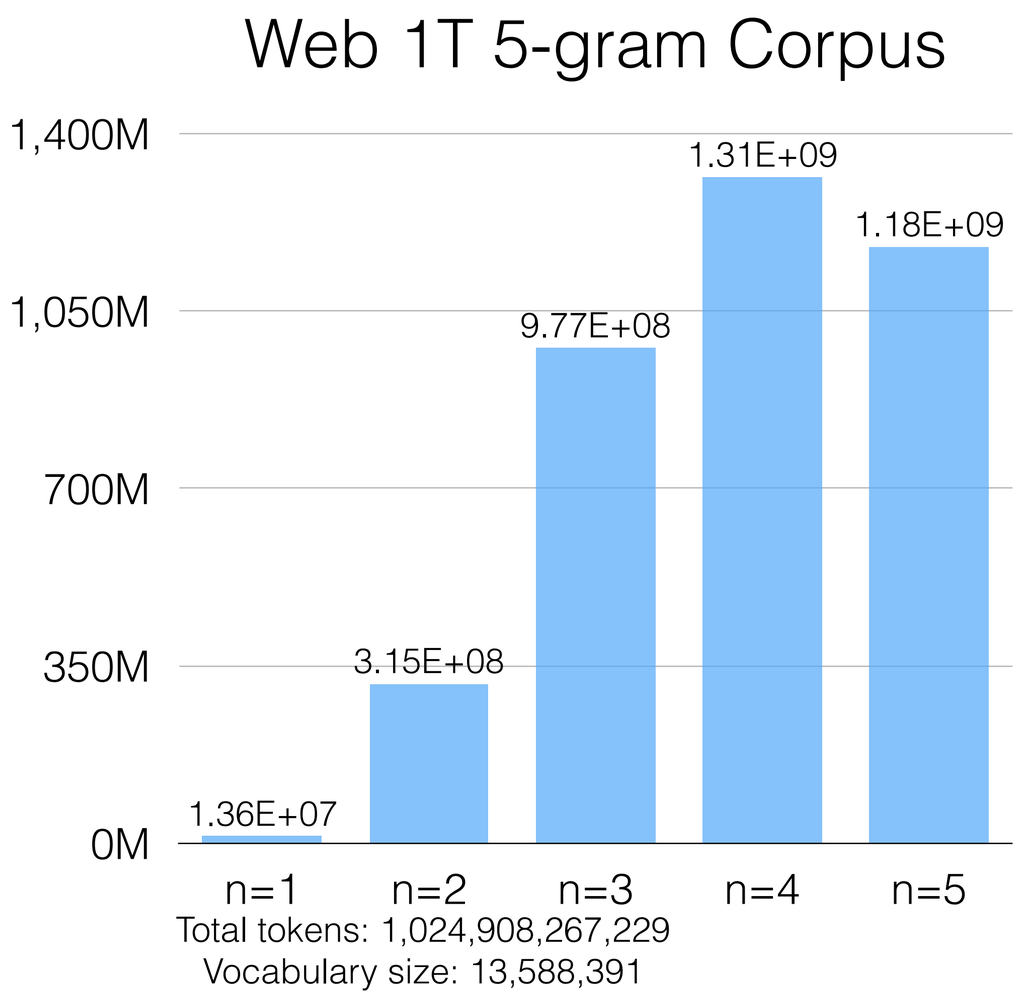
Trigram Sentence Probability

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

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.