Inspire…Educate…Transform.

1. Language Modeling

Dr. Manish Gupta

Sr. Mentor – Academics, INSOFE

Adapted from http://web.stanford.edu/class/cs276a/handouts/lecture12.pdf
ChengXiang Zhai. Statistical Language Models for Information Retrieval. Synthesis Lectures on Human Language Technologies. 2008. Morgan & Claypool Publishers.

Course Content

- Collection of three main topics of high recent interest.
  - Search engines (Crawling, Indexing, Ranking)
    - **Language Modeling**
    - Text Indexing and Crawling
    - Relevance Ranking
    - Link Analysis Algorithms
  - Text Processing (NLP, NER, Sentiments)
    - Natural Language Processing
    - Named Entity Recognition
    - Sentiment Analysis
    - Summarization
  - Social networks (Properties, Influence Propagation)
    - Social Network Analysis
    - Influence Propagation in Social Networks

Relationship with Other Courses

- Fundamentals of Probability and Statistical Methods
  - These are the basics. You need them to understand the theoretical background for the course.
- Advanced Machine Learning, and Methods and Algorithms in Machine Learning
  - Lots of tasks in the course use ML algorithms and classifiers.
- Engineering Big Data with R and Hadoop Ecosystem
  - The tasks that we will discuss, especially ones related to search engines can’t be done on single machines.
  - We need Big data systems as the basic infrastructure for running the search engine algorithms.

Agenda

- Essential Probability and Statistics

Prob/Statistics and Text Management

- Probability and statistics provide a principled way to quantify the uncertainties associated with natural language
- Allow us to answer questions like
  - Given that we observe “baseball” three times and “game” once in a news article, how likely is it about “sports”? (text categorization, information retrieval)
  - Given that a user is interested in sports news, how likely would the user use “baseball” in a query? (information retrieval)

Basic Concepts in Probability (1)

- Random experiment: an experiment with uncertain outcome (e.g., tossing a coin, picking a word from text)
- Sample space: all possible outcomes, e.g.,
  - Tossing 2 fair coins, S ={HH, HT, TH, TT}
- Event: $E \subseteq S$, E happens iff outcome is in E, e.g.,
  - E={HH} (all heads)
  - E={HH,TT} (same face)
  - Impossible event ({}), certain event (S)
- Probability of Event : $1 \geq P(E) \geq 0$, s.t.
  - P(S)=1 (outcome always in S)
  - $P(A \cup B)=P(A)+P(B)$ if $(A \cap B)=\emptyset$ (e.g., A=same face, B=different face)

Basic Concepts in Probability (2)

- Joint probability: $P(A,B,C)$ or $P(A \cap B \cap C)$
- Conditional Probability : $P(B|A)=P(A \cap B)/P(A)$
  - $P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$
  - So, $P(A|B)=P(B|A)P(A)/P(B)$ (Bayes' Rule)
  - For independent events, $P(A \cap B) = P(A)P(B)$, so $P(A|B)=P(A)$
  - Conditional Independence: $P(A \cap B|C) = P(A|C)\, P(B|C)$
- Total probability: If $A_1, \ldots, A_n$ form a partition of S, then
  - $P(B)= P(B \cap S)=P(B \cap A_1)+\ldots+P(B \cap A_n)$
  - So, $P(A_i|B)=P(B|A_i)P(A_i)/P(B)$

    $= P(B|A_i)P(A_i)/[P(B|A_1)P(A_1)+\ldots+P(B|A_n)P(A_n)]$
  - This allows us to compute $P(A_i|B)$ based on $P(B|A_i)$

Agenda

- Essential Probability and Statistics
- N-gram Models of Language

Next Word Prediction

- From a NY Times story...
  - Stocks ...
  - Stocks plunged this ....
  - Stocks plunged this morning, despite a cut in interest rates
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began

Next Word Prediction

- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last …
- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last Tuesday's terrorist attacks.

Human Word Prediction

- Clearly, at least some of us have the ability to predict future words in an utterance.
- How?
  - Domain knowledge: red blood vs. red hat
  - Syntactic knowledge: the…<adj|noun>
  - Lexical knowledge: baked <potato vs. chicken>
- A useful part of the knowledge needed to allow Word Prediction can be captured using simple statistical techniques
- In particular, we'll be interested in the notion of the probability of a sequence (of letters, words,…)

Applications of Word Prediction

- Why do we want to predict a word, given some preceding words?
  - Rank the likelihood of sequences containing various alternative hypotheses
  - Assess the likelihood/goodness of a sentence
- Spelling checkers
  - They are leaving in about fifteen **minuets** to go to her house
  - The study was conducted mainly **be** John Black.
  - I need to **notified** the bank of this problem.
- Mobile phone texting
  - He is trying to **fine** out.
  - Hopefully, all **with** continue smoothly in my absence.
- Speech recognition
  - Theatre owners say **popcorn/unicorn** sales have doubled...
- Handwriting recognition
- Disabled users
- Machine Translation

How to do Word Prediction?

- Use the previous N-1 words in a sequence to predict the next word
- Language Model (LM)
  - unigrams, bigrams, trigrams,…
- How do we train these models?
  - Very large corpora
  - Corpora are online collections of text and speech
    - Wall Street Journal
    - Newswire

N-grams

- Assume a language has V unique words in its lexicon, how likely is word x to follow word y?
  - Simplest model of word probability: 1/V
  - Alternative 1: estimate likelihood of x occurring in new text based on its general frequency of occurrence estimated from a corpus (unigram probability)

    popcorn is more likely to occur than unicorn
  - Alternative 2: condition the likelihood of x occurring in the context of previous words (bigrams, trigrams,…)

    mythical unicorn is more likely than mythical popcorn

Prob of a Word Sequence

- By the Chain Rule we can decompose a joint probability, e.g. P(w_1,w_2,w_3) as follows
  - $P(w_1^n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$
- Compute the product of component conditional probabilities
  - P(the mythical unicorn) = P(the) * P(mythical|the) * P(unicorn|the mythical)
- But…the *longer* the sequence, the *less likely* we are to find it in a training corpus

  P(Most biologists and folklore specialists believe that in fact the mythical unicorn horns derived from the narwhal)
- What can we do?

Bigram Model

- Approximate $P(w_n|w_1^{n-1})$ by $P(w_n|w_{n-1})$
  - E.g., P(unicorn|the mythical) by P(unicorn|mythical)
- Markov *assumption*: the probability of a word depends only on the probability of a limited history
- ***Generalization: the probability of a word depends only on the probability of the n previous words***
  - trigrams, 4-grams, 5-grams,…
  - the higher n is, the more data needed to train
  - backoff models…

Andrei Markov

Using N-grams

- For N-gram models
  - $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$
    - E.g. for bigrams, $P(w_8|w_1^{8-1}) \approx P(w_8|w_{8-2+1}^{8-1})$
  - By the Chain Rule we can decompose a joint probability, e.g. $P(w_1,w_2,w_3)$ as follows
    - $P(w_1,w_2, \ldots,w_n) = P(w_n|w_{n-1},w_{n-2},\ldots,w_1)\ P(w_{n-1}|w_{n-2}, \ldots,w_1) \ldots P(w_2|w_1)\ P(w_1)$
  - Hence, $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1})$
  - For bigrams then, the probability of a sequence is just the product of the conditional probabilities of its bigrams
    - E.g., P(the,mythical,unicorn) = P(unicorn|mythical) P(mythical|the)

Training and Testing

- N-Gram probabilities come from a training corpus
  - overly narrow corpus: probabilities don't generalize
  - overly general corpus: probabilities don't reflect task or domain
- A separate test corpus is used to evaluate the model
- A simple bigram example
  - Estimate the likelihood of the sentence I want to eat Chinese food.
    - P(I want to eat Chinese food) = P(I | <start>) P(want | I) P(to | want) P(eat | to) P(Chinese | eat) P(food | Chinese) P(<end>|food)
  - What do we need to calculate these likelihoods?
    - Bigram probabilities for each word pair sequence in the sentence
    - Calculated from a large corpus

Sample Bigram Probabilities

Eat on	.16	Eat Thai	.03
Eat some	.06	Eat breakfast	.03
Eat lunch	.06	Eat in	.02
Eat dinner	.05	Eat Chinese	.02
Eat at	.04	Eat Mexican	.02
Eat a	.04	Eat tomorrow	.01
Eat Indian	.04	Eat dessert	.007
Eat today	.03	Eat British	.001

<start> I	.25	Want some	.04
<start> I'd	.06	Want Thai	.01
<start> Tell	.04	To eat	.26
<start> I'm	.02	To have	.14
I want	.32	To spend	.09
I would	.29	To be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
Want to	.65	British cuisine	.01
Want a	.05	British lunch	.01

Text Generation with Bigram Model

- P(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British) = .25*.32*.65*.26*.001*.60 = .000080
  - How would we calculate I want to eat Chinese food ?
- Probabilities roughly capture "syntactic" facts and "world knowledge"
  - eat is often followed by an NP
  - British food is not too popular
    - "eat British" has lower prob than "eat Chinese"
    - "British food" has lower prob than "Chinese food"

Approximating Shakespeare

- Generating sentences with random unigrams...
  - Every enter now severally so, let
  - Hill he late speaks; or! a more to leg less first you enter
- With bigrams...
  - What means, sir. I confess she? then all sorts, he is trim, captain.
  - Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry.
- Trigrams
  - Sweet prince, Falstaff shall die.
  - This shall forbid it should be branded, if renown made it empty.

Approximating Shakespeare

- Quadrigrams
  - What! I will go seek the traitor Gloucester.
  - Will you not tell me who I am?
  - What's coming out here looks like Shakespeare because it ***is*** Shakespeare
- Note: ***As we increase the value of N, the accuracy of an n-gram model increases, since choice of next word becomes increasingly constrained***

N-gram Training Sensitivity

- If we repeated the Shakespeare experiment but trained our n-grams on a Wall Street Journal corpus, what would we get?
- Note: ***This question has major implications for corpus selection or design***
- WSJ is ***not*** Shakespeare: Sentences Generated from WSJ
  - unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
  - bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
  - trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Observations from Shakespeare Corpus

- There are 884,647 tokens, with 29,066 unique words, in an approximately one million word Shakespeare corpus
  - Shakespeare produced 300,000 unique bigrams out of 844 million possible bigrams: so, ***99.96% of the possible bigrams were never seen (have zero entries in the table)***
- A small number of events occur with high frequency
- A large number of events occur with low frequency
- You can quickly collect statistics on the high frequency events
- You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Some zeroes in the table are really zeros But others are simply low frequency events you haven't seen yet. How to address?
  - Using smoothing
  - Using backoff method
  - Using Interpolation method

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram

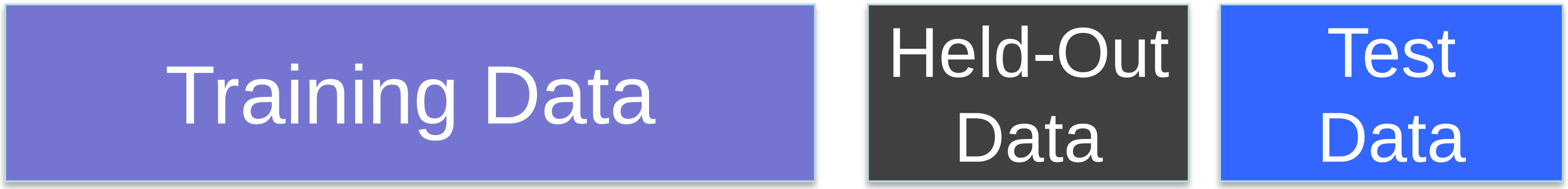
$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

- Interpolation works better

How to set the lambdas?

- Use a **held-out** corpus

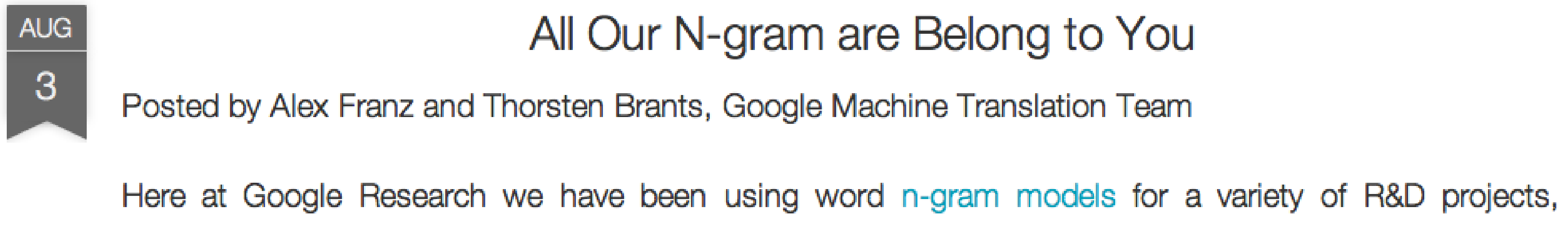


- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid \lambda_1 \dots \lambda_k) = \sum_i \log P_{\lambda_1 \dots \lambda_k}(w_i \mid w_{i-1})$$

Google N-Gram Release, August 2006

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html



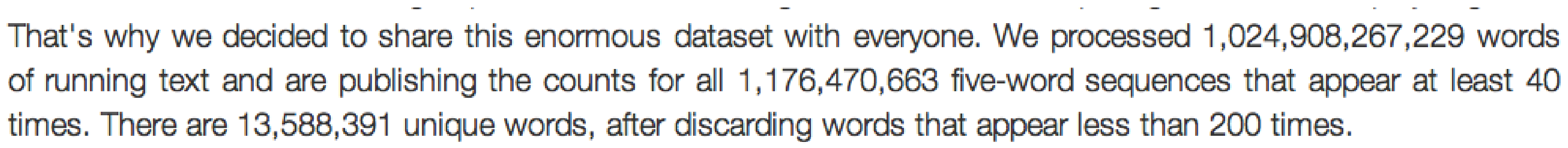
AUG
3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

…



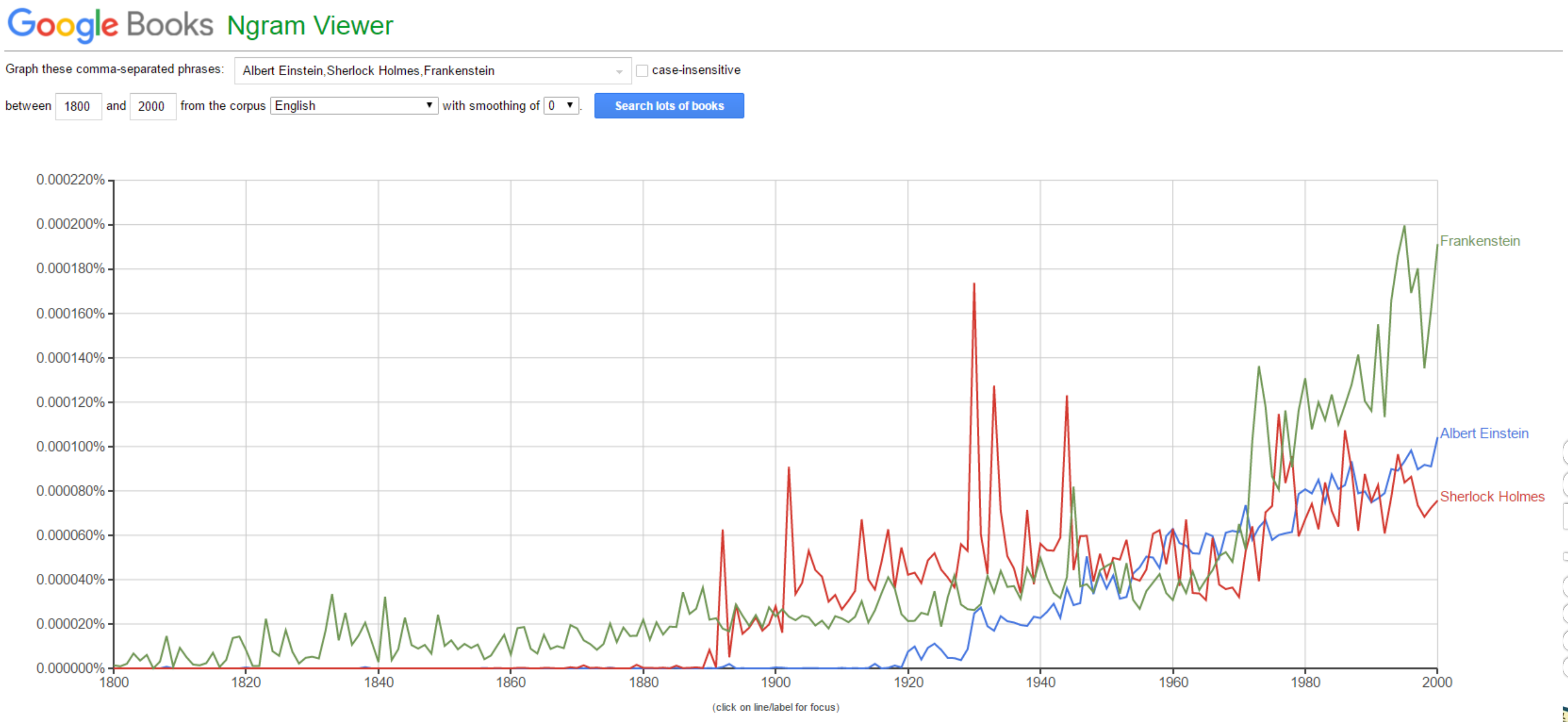
That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

Google Book N-grams Viewer

- http://ngrams.googlelabs.com/



Google Books Ngram Viewer

Graph these comma-separated phrases: Albert Einstein,Sherlock Holmes,Frankenstein ☐ case-insensitive

between 1800 and 2000 from the corpus English with smoothing of 0.

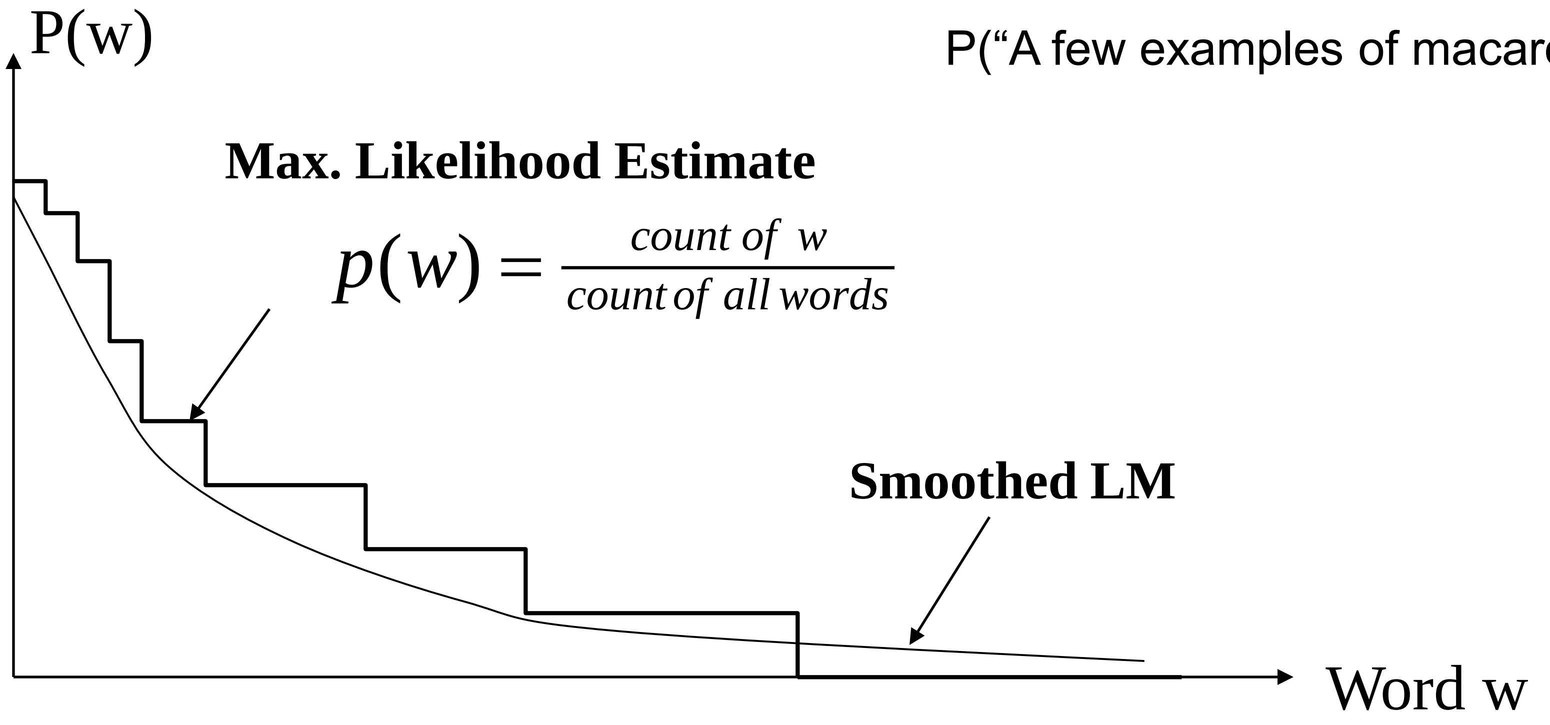
Agenda

- Essential Probability and Statistics
- N-gram Models of Language
- Smoothing

Smoothing

- Words follow a Zipfian distribution
  - Small number of words occur very frequently. A large number are seen only once.
  - Zipf's law: *a word's frequency is approximately inversely proportional to its rank in the word distribution list*
- Zero probabilities on one bigram cause a zero probability on the entire sentence. So….how do we estimate the likelihood of unseen n-grams?

P("A few examples of macaronic songs")



$$p(w) = \frac{count\ of\ w}{count\ of\ all\ words}$$

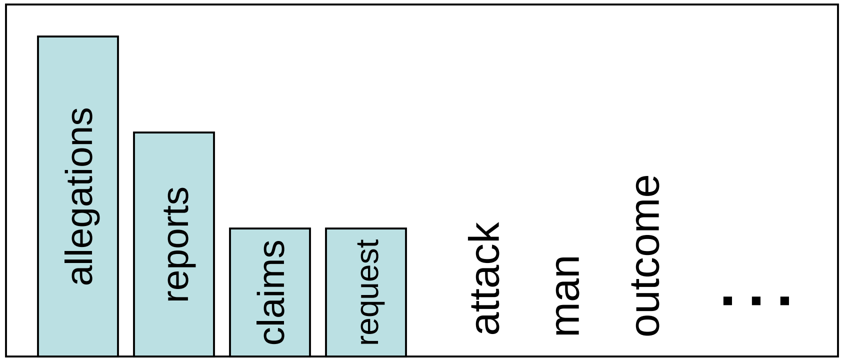
Smoothing: Robin Hood

- Steal from the rich and give to the poor (in probability mass)
- All smoothing methods try to
  - discount the probability of words seen in a doc
  - re-allocate the extra probability so that unseen words will have a non-zero probability
- When we have sparse statistics:

P(w | denied the)

3 allegations
2 reports
1 claims
1 request

7 total

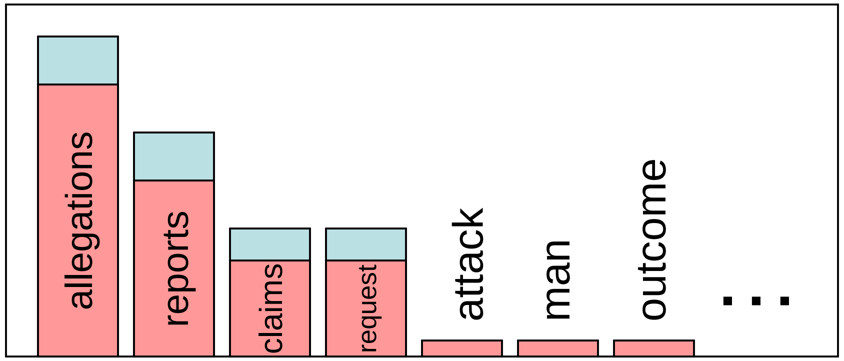


- Steal probability mass to generalize better

P(w | denied the)

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



Laplace Smoothing

- For unigrams:
  - Add 1 to every word (type) count to get an adjusted count c^*
  - Normalize by N (#tokens) + V (#unique words)
  - Original unigram probability
    - $P(w_i) = \frac{c_i}{N}$
  - New unigram probability
    - $P(w_i) = \frac{c_i+1}{N+V}$

- Tiny Corpus, V=4; N=20

Word	True Count	Unigram Prob	New Count	Adjusted Prob
eat	10	.5	11	.46
British	4	.2	5	.21
food	6	.3	7	.29
happily	0	.0	1	.04
	20	1.0	~20	1.0

Laplace Smoothing

- *So, we lower some (larger) observed counts in order to include unobserved vocabulary*
- For bigrams:
  - Original
    - $P(w_i|w_{i-1}) = \frac{c(w_i|w_{i-1})}{c(w_{i-1})}$
  - New
    - $P(w_i|w_{i-1}) = \frac{c(w_i|w_{i-1})+1}{c(w_{i-1})+V}$
  - *But this changes counts drastically*:
    - *Too much weight* given to unseen ngrams
    - In practice, unsmoothed bigrams often work better!

Agenda

- Essential Probability and Statistics
- N-gram Models of Language
- Smoothing
- Evaluation and Perplexity

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Lower Perplexity = Better Model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Take-aways

- Probability is useful in text processing also.
- N-gram models are very useful for information retrieval.
- Smoothing helps us avoid zero probability issues.
- Perplexity is a good measure for intrinsic evaluation of n-gram models.

References

- ChengXiang Zhai. Statistical Language Models for Information Retrieval. Synthesis Lectures on Human Language Technologies. 2008. Morgan & Claypool Publishers.
- http://web.stanford.edu/class/cs276a/handouts/lecture12.pdf
- *Introduction Speech and Language Processing* – Jurafsky and Martin

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals:	+91-9502334561/63 or 040-65743991
For Corporates:	+91-9618483483
Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering

Another Reason for Smoothing

Content words

	Query = “the	algorithms	for	data	mining”
p(w|d1):	*0.04*	*0.001*	*0.02*	*0.002*	*0.003*
p(w|d2):	*0.02*	*0.001*	*0.01*	*0.003*	*0.004*

p(“algorithms”|d1) = p(“algorithms”|d2)
p(“data”|d1) < p(“data”|d2)
p(“mining”|d1) < p(“mining”|d2)

$\Rightarrow$ Intuitively, d2 should have a higher score, but p(q|d1)>p(q|d2)…

So we should make p(“the”) and p(“for”) less different for all docs, and smoothing helps achieve this goal… $p_{smoothed}(q|d1) > p_{smoothed}(q|d2)$

After smoothing: $p_{smoothed}(w|d)$=0.1p(w|d)+0.9p(w|Collection)

Query	= “the	algorithms	for	data	mining”
P(w|collection)	0.2	0.00001	0.2	0.00001	0.00001
$p_{smoothed}(w\|d1)$:	0.184	0.000109	0.182	0.000209	0.000309
$p_{smoothed}(w\|d2)$:	0.182	0.000109	0.181	0.000309	0.000409

Dirichlet Smoothing

- $P(t|d) = \frac{tf_{t,d}+\alpha P(t|M_C)}{L_d+\alpha} = \frac{L_d}{L_d+\alpha} P(t|M_d) + \frac{\alpha}{L_d+\alpha} P(t|M_C)$
- The background distribution P(t|M_c) is the prior for P(t|d).
- Intuition: Before having seen any part of the document we start with the background distribution as our estimate. As we read the document and count terms we update the background distribution.
- The weighting factor α determines how strong an effect the prior has.
- Jelinek Mercer vs 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.

Two-stage Smoothing

Stage-1

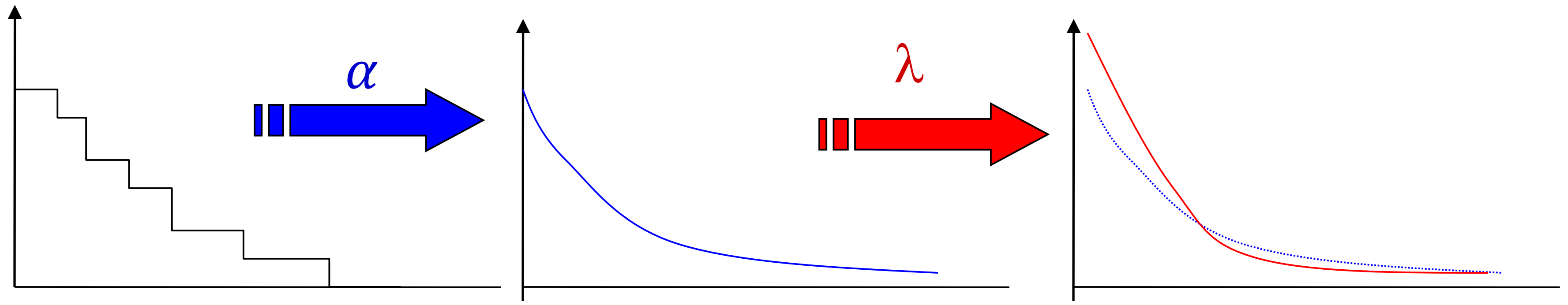
-Explain unseen words

-Dirichlet prior (Bayesian)

Stage-2

-Explain noise in query (for, the)

-2-component mixture



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

User background model (noise)

λ and α can be automatically set through statistical estimation

Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell
- Use the count of things we've **seen once**
  - to help estimate the count of things we've **never seen**