Università di Pisa

Language Modeling

Giuseppe Attardi

Text Analytics

IP notice: some slides from: Dan Jurafsky, Jim Martin, Sandiway Fong, Dan Klein

Outline

- Language Modeling (N-grams)
  - N-gram Intro
  - The Chain Rule
  - The Shannon Visualization Method
  - Evaluation:
    - Perplexity
  - Smoothing:
    - Laplace (Add-1)
    - Add-prior

Probabilistic Language Model

- Goal: assign a probability to a sentence
- Machine Translation:
  - P(high winds tonite) > P(large winds tonite)
- Spell Correction
  - “The office is about fifteen minuets from my house"
    - P(about fifteen minutes from) > P(about fifteen minuets from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- Summarization, question--answering, etc.

Why Language Models

- We have an English speech recognition system, which answer is better?

Speech

Interpretation

speech recognition system

speech cognition system

speck podcast histamine

スピーチ が 救出 ストン

- Language models tell us the answer!

Language Modeling

- We want to compute

  $P(w_1,w_2,w_3,w_4,w_5 \ldots w_n) = P(W)$

  = the probability of a sequence
- Alternatively we want to compute

  $P(w_5|w_1,w_2,w_3,w_4)$

  = the probability of a word given some previous words
- The model that computes

  $P(W)$ or

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

  is called the language model.
- A better term for this would be "The Grammar"
- But "Language model" or LM is standard

Computing $P(W)$

- How to compute this joint probability:

  P(“the”, “other”, “day”, “I”, “was”, “walking”, “along”, “and”, “saw”, “a”, “lizard”)

- Intuition: let’s rely on the Chain Rule of Probability

The Chain Rule

- Recall the definition of conditional probabilities

$$P(B \mid A) = \frac{P(A \wedge B)}{P(A)}$$

- Rewriting:

$$P(A \wedge B) = P(A)P(B \mid A)$$

- More generally

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- In general

$$P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n\text{-}1})$$

The Chain Rule applied to joint probability of words in sentence

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

P("the big red dog was") =

P(the) • P(big|the) • P(red|the big) • P(dog|the big red) • P(was|the big red dog)

Obvious estimate

- How to estimate?

  $P(the \mid its\ water\ is\ so\ transparent\ that)$

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

Unfortunately

- There are a lot of possible sentences
- We will never be able to get enough data to compute the statistics for those long prefixes

$$P(lizard|the,other,day,I,was,walking,along,and,saw,a)$$

or

$$P(the|its\ water\ is\ so\ transparent\ that)$$

Markov Assumption

- Make the simplifying assumption

  $P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)$

- or maybe

  $P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)$

Markov Assumption

- So for each component in the product, replace with the approximation (assuming a prefix of N)

$$P(w_n|w_1^{n-1}) \simeq P(w_n|w_{n-N+1}^{n-1})$$

- Bigram model

$$P(w_n|w_1^{n-1}) \simeq P(w_n|w_{n-1})$$

N-gram models

- We can extend to trigrams, 4--grams, 5--grams
- In general this is an insufficient model of language
  - because language has long--distance dependencies:
  - “The computer which I had just put into the machine room on the fifth floor crashed.”
- But we can often get away with N--gram models

Estimating bigram probabilities

The Maximum Likelihood Estimate

$$P(w_n|w_1^{n-1}) \simeq \frac{count(w_{n-N+1}^{n-1}, w_n)}{count(w_{n-N+1}^{n-1})}$$

$$P(w_n|w_1^{n-1}) \simeq \frac{count(w_{n-1}, w_n)}{count(w_{n-1})}$$

An example

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$$

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

This is the *Maximum Likelihood Estimate*, because it is the one which maximizes *P(Training set|Model)*

Maximum Likelihood Estimates

- The **Maximum Likelihood Estimate** of some parameter of a model *M* from a training set *T*
  - is the estimate that
  - maximizes the likelihood of the training set *T* given the model *M*
- Suppose the word “Chinese” occurs 400 times in a corpus of a million words (e.g. the Brown corpus)
- What is the probability that a random word from some other text will be “Chinese”
- MLE estimate is 400/1000000 = .004
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that “Chinese” will occur 400 times in a million word corpus.

Maximum Likelihood

We want to estimate the probability, p, that individuals are infected with a certain kind of parasite.

Ind.	Infected	Probability of observation
1	1	p
2	0	$1-p$
3	1	p
4	1	p
5	0	$1-p$
6	1	p
7	1	p
8	0	$1-p$
9	0	$1-p$
10	1	p

The maximum likelihood method (discrete distribution):

1. Write down the probability of each observation by using the model parameters
2. Write down the probability of all the data

$$\Pr(\text{Data} \mid p) = p^6(1-p)^4$$

3. Find the value parameter(s) that maximize this probability

Maximum likelihood

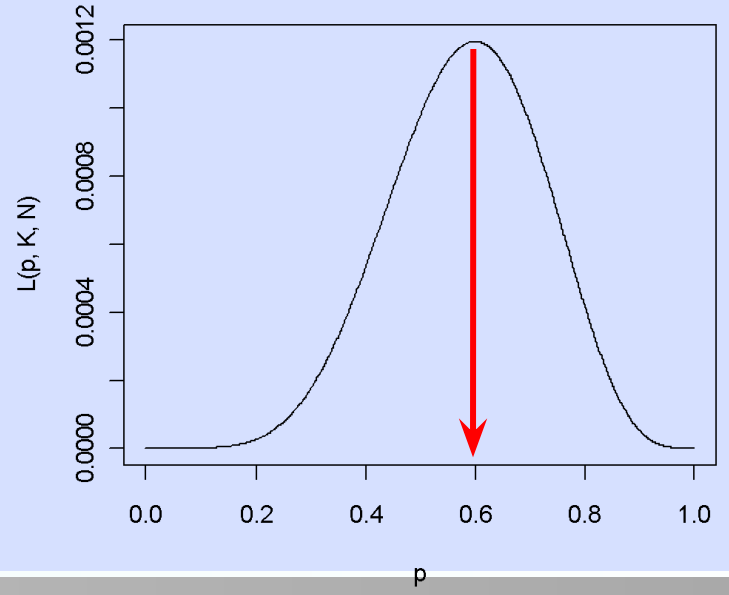
We want to estimate the probability, p, that individuals are infected with a certain kind of parasite.

Ind.	Infected	Probability of observation
1	1	p
2	0	$1-p$
3	1	p
4	1	p
5	0	$1-p$
6	1	p
7	1	p
8	0	$1-p$
9	0	$1-p$
10	1	p

Likelihood function:

$$L(p) = \Pr(\text{Data} \mid p) = p^6(1-p)^4$$

- Find the value parameter(s) that maximize this probability



Computing the MLE

- Set the derivative to 0:

$$0 = \frac{d}{dp} p^6 (1-p)^4 =$$

$$6p^5(1-p)^4 - p^6 4(1-p)^3 =$$

$$p^5(1-p)^3[6(1-p) - 4p) =$$

$$p^5(1-p)^3[6 - 10p]$$

- Solutions:
  - p = 0 (minimum)
  - p = 1 (minimum)
  - p = 0.6 (maximum)

More examples: Berkeley Restaurant Project

can you tell me about any good cantonese restaurants close by

mid priced thai food is what i'm looking for

tell me about chez panisse

can you give me a listing of the kinds of food that are available

i'm looking for a good place to eat breakfast

when is caffe venezia open during the day

Raw bigram counts

Out of 9222 sentences

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

Raw bigram probabilities

Normalize by unigrams (divide by $C(w_{-1})$):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

$$P(\langle s\rangle\ I\ want\ english\ food\ \langle/s\rangle) =$$
$$P(i|\langle s\rangle)\ x$$
$$P(want|I)\ x$$
$$P(english|want)\ x$$
$$P(food|english)\ x$$
$$P(\langle/s\rangle|food)$$
$$=.000031$$

What kinds of knowledge?

$$P(english|want) = .0011$$
$$P(chinese|want) = .0065$$
$$P(to|want) = .66$$
$$P(eat \mid to) = .28$$
$$P(food \mid to) = 0$$
$$P(want \mid spend) = 0$$
$$P(i \mid <s>) = .25$$

Practical Issues

- Compute in log space
  - Avoid underflow
  - Adding is faster than multiplying

$$\log(p_1 \bullet p_2 \bullet p_3 \bullet p_4) = \log(p_1) + \log(p_2) + \log(p_3) + \log(p_4)$$

Shannon's Game

- What if we turn these models around and use them to *generate* random sentences that are *like* the sentences from which the model was derived.

The Shannon Visualization Method

- Generate random sentences:
- Choose a random bigram <s>, w according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

<s> I
I want
want to
to eat
eat Chinese
Chinese food
food </s>

Approximating Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

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. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams: so, 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams:
  - What's coming out looks like Shakespeare because it ***is*** Shakespeare

The Wall Street Journal is not Shakespeare (no offense)

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

Lesson 1: the perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models, adapt to test set, etc.

Train and Test Corpora

- A language model must be trained on a large corpus of text to estimate good parameter values.
- Model can be evaluated based on its ability to predict a high probability for a disjoint (held-out) test corpus (testing on the training corpus would give an optimistically biased estimate).
- Ideally, the training (and test) corpus should be representative of the actual application data.
- May need to *adapt* a general model to a small amount of new (*in-domain*) data by adding highly weighted small corpus to original training data.

Smoothing

Smoothing

- Since there are a combinatorial number of possible word sequences, many rare (but not impossible) combinations never occur in training, so MLE incorrectly assigns zero to many parameters (aka *sparse data*).
- If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero (i.e. infinite perplexity).
- In practice, parameters are *smoothed* (aka *regularized*) to reassign some probability mass to unseen events.
  - Adding probability mass to unseen events requires removing it from seen ones (*discounting*) in order to maintain a joint distribution that sums to 1.

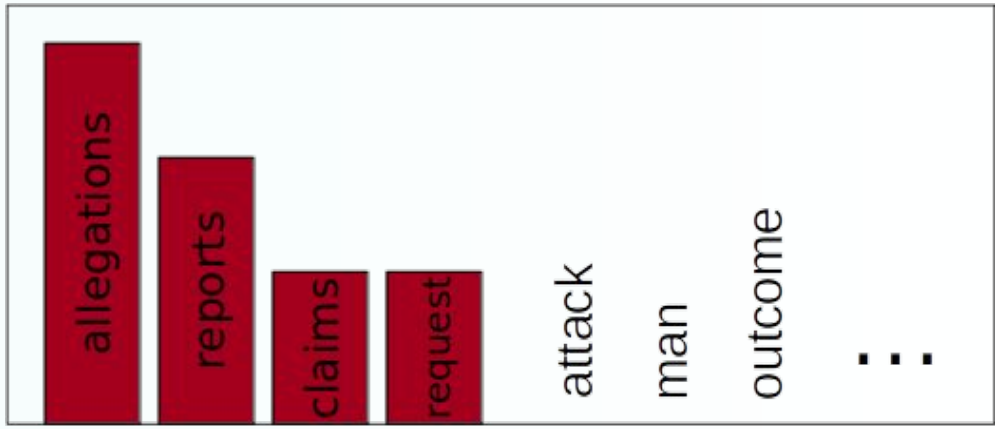
Smoothing is like Robin Hood:

Steal from the rich and give to the poor (in probability mass)

- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request

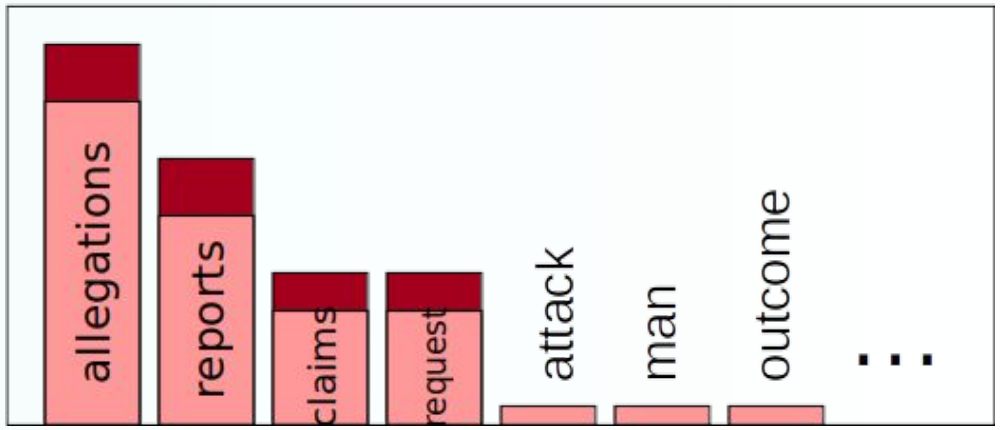
  7 total



- 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



Slide from Dan Klein

Laplace smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple
- MLE estimate:

$$P(w_i) = \frac{c_i}{N}$$

- Laplace estimate:

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Reconstructed counts:

$$c_i^* = (c_i + 1)\frac{N}{N + V}$$

Laplace smoothed bigram counts

Berkeley Restaurant Corpus

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Note big change to counts

- $C(want\ to)$ went from 608 to 238!
- $P(to|want)$ from 0.66 to 0.26!
- Discount $d = c^*/c$
  - d for "chinese food" = 0.10 A 10x reduction!
  - So in general, Laplace is a blunt instrument
- But Laplace smoothing not used for N-grams, as we have much better methods
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge.

Add-k

- Add a small fraction instead of 1
- k = 0.01

Even better: Bayesian unigram prior smoothing for bigrams

- Maximum Likelihood Estimation

$$P(w_2 \mid w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

- Laplace Smoothing

$$P_{Laplace}(w_2 \mid w_1) = \frac{C(w_1, w_2) + 1}{C(w_1) + vocal}$$

- Bayesian Prior Smoothing

$$P_{Prior}(w_2 \mid w_1) = \frac{C(w_1, w_2) + P(w_2)}{C(w_1) + 1}$$

Lesson 2: zeros or not?

- Zipf's Law:
  - 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
- Result:
  - Our estimates are sparse! no counts at all for the vast bulk of things we want to estimate!
  - Some of the zeroes in the table are really zeros But others are simply low frequency events you haven't seen yet. After all, ANYTHING CAN HAPPEN!
  - How to address?
- Answer:
  - Estimate the likelihood of unseen N-grams!

Slide from B. Dorr and J. Hirschberg

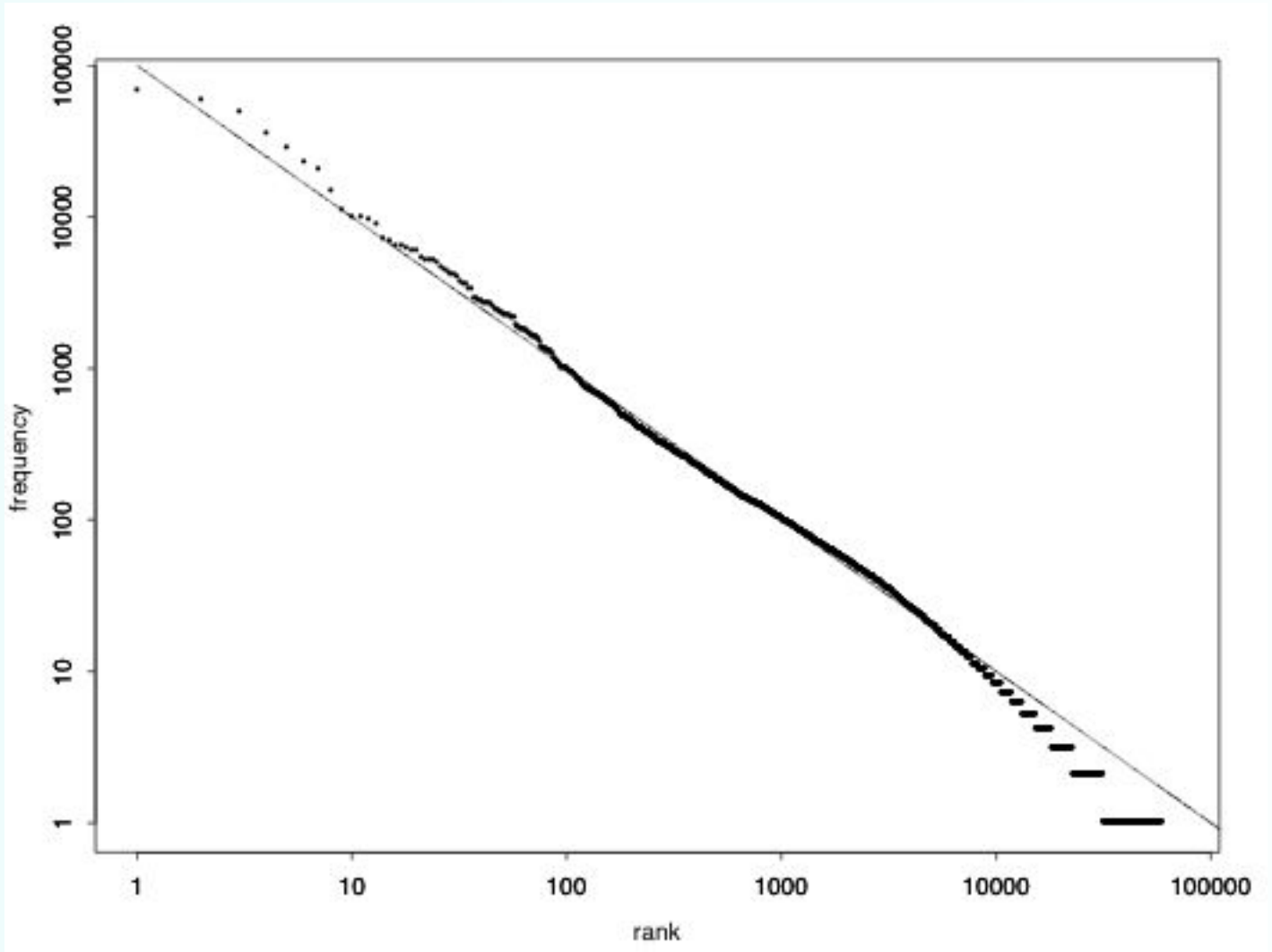
Zipf's law

Word	Freq. (f)	Rank (r)	$f \cdot r$
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
about	158	60	9480
more	138	70	9660
never	124	80	9920
Oh	116	90	10440
two	104	100	10400

Word	Freq. (f)	Rank (r)	$f \cdot r$
turned	51	200	10200
you'll	30	300	9000
name	21	400	8400
comes	16	500	8000
group	13	600	7800
lead	11	700	7700
friends	10	800	8000
begin	9	900	8100
family	8	1000	8000
brushed	4	2000	8000
sins	2	3000	6000
Could	2	4000	8000
Applausive	1	8000	8000

$f \propto 1/r$ (f proportional to 1/r)
there is a constant k such that
$f \cdot r = k$

Zipf's Law for the Brown Corpus



Zipf law: interpretation

- Principle of *least effort*: both the speaker and the hearer in communication try to minimize effort:
  - Speakers tend to use a small vocabulary of common (shorter) words
  - Hearers prefer a large vocabulary of rarer less ambiguous words
  - Zipf's law is the result of this compromise
- Other laws …
  - Number of meanings *m* of a word obeys the law: $m \propto 1/\sqrt{f}$
  - Inverse relationship between frequency and length

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Language Modeling Toolkits

- SRILM
  http://www.speech.sri.com/projects/srilm/
- IRSTLM
- Ken LM

Google N-Gram Release

All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

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, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training

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 Book N-grams

- http://ngrams.googlelabs.com/

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

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

Evaluation and Perplexity

Evaluation

- Train parameters of our model on a **training set**.
- How do we evaluate how well our model works?
- Look at the models performance on some new data
- This is what happens in the real world; we want to know how our model performs on data we haven't seen
- Use a **test set**. A dataset which is different than our training set
- Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
- One such metric is **perplexity**

Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a task (language identification, speech recognizer, machine translation system)
  - Run the task, get an accuracy for A (how many langs identified correctly, or Word Error Rate, or etc)
  - Put model B in task, get accuracy for B
  - Compare accuracy for A and B
  - **Extrinsic evaluation**

Language Identification task

- Create an N-gram model for each language
- Compute the probability of a given text

$$P_{lang1}(text)$$
$$P_{lang2}(text)$$
$$P_{lang3}(text)$$

- Select language with highest probability

$$lang = argmax_l P_l(text)$$

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **just** like the training data
  - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**

Perplexity

- The intuition behind perplexity as a measure is the notion of surprise.
- How surprised is the language model when it sees the test set?
  - Where surprise is a measure of...
    - Gee, I didn't see that coming...
  - The more surprised the model is, the lower the probability it assigned to the test set
  - The higher the probability, the less surprised it was

Perplexity

- Measures of how well a model “fits” the test data.
- Uses the probability that the model assigns to the test corpus.
- Normalizes for the number of words in the test corpus and takes the inverse.

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

- Measures the weighted average branching factor in predicting the next word (lower is better).

Perplexity

- Perplexity:

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

- Chain rule:

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

- For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

Perplexity as branching factor

- Let's suppose a sentence consists of random digits
- How hard is the task of recognizing digits '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'
- Perplexity: 10

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Lower perplexity = better model

- Model trained on 38 million words from the Wall Street Journal (WSJ) using a 19,979 word vocabulary.
- Evaluation on a disjoint set of 1.5 million WSJ words.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unknown Words

- How to handle words in the test corpus that did not occur in the training data, i.e. *out of vocabulary* (OOV) words?
- Train a model that includes an explicit symbol for an unknown word (<UNK>):
  1. Choose a vocabulary in advance and replace other words in the training corpus with <UNK>, or
  2. Replace the first occurrence of each word in the training data with <UNK>.

Unknown Words handling

- Training of <UNK> probabilities
  - Create a fixed lexicon L of size V
  - Any training word not in L changed to <UNK>
  - Now we train its probabilities like a normal word
- At decoding time
  - In text input: use <UNK> probabilities for any word not in training

Smoothing

Advanced LM stuff

- Current best smoothing algorithm
  - Kneser-Ney smoothing
- Other stuff
  - Interpolation
  - Backoff
  - Variable-length n-grams
  - Class-based n-grams
    - Clustering
    - Hand-built classes
  - Cache LMs
  - Topic-based LMs
  - Sentence mixture models
  - Skipping LMs
  - Parser-based LMs
  - Word Embeddings

discount

$$p_{KN}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - \delta, 0)}{\sum_{w'} c(w_{i-1}, w')} + \lambda_{w_{i-1}} p_{KN}(w_i)$$

how likely it is to see the word w_i in an unfamiliar context

Backoff and Interpolation

If we are estimating:

- Trigram $P(z|xy)$
- but $C(xyz)$ is zero

Use info from:

- Bigram $P(z|y)$

Or even:

- Unigram $P(z)$

How to combine the trigram/bigram/unigram info?

Backoff versus interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation**: mix all three

Backoff

- Only use lower-order model when data for higher-order model is unavailable
- Recursively back-off to weaker models until data is available

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n}) > 1 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

Where P^* is a discounted probability estimate to reserve mass for unseen events and α's are back-off weights (see book for details).

Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus

- Choose lambdas which maximize the probability of data
  i.e. fix the N-gram probabilities
  then search for lambda values that,
  when plugged into previous equation,
  give largest probability for held-out set
  Can use EM (Expectation Maximization) to do this search

Intuition of backoff+discounting

- How much probability to assign to all the zero trigrams?
  - Use Good-Turing or other discounting algorithm
- How to divide that probability mass among different contexts?
  - Use the N-1 gram estimates
- What do we do for the unigram words not seen in training?
  - **Out Of Vocabulary** = OOV words

Problem for N-Grams: Long Distance Dependencies

- Sometimes local context does not provide enough predictive clues, due to the presence of *long-distance dependencies*.
  - Syntactic dependencies
    - "The *man* next to the large oak tree near the grocery store on the corner is tall."
    - "The *men* next to the large oak tree near the grocery store on the corner are tall."
  - Semantic dependencies
    - "The *bird* next to the large oak tree near the grocery store on the corner flies rapidly."
    - "The *man* next to the large oak tree near the grocery store on the corner talks rapidly."
- More complex models of language are needed to handle such dependencies.

ARPA format

unigram:	$\log p^*(w_i)$	w_i	$\log \alpha(w_i)$
bigram:	$\log p^*(w_i \mid w_{i-1})$	$w_{i-1}w_i$	$\log \alpha(w_{i-1}w_i)$
trigram:	$\log p^*(w_i \mid w_{i-2}, w_{i-1})$	$w_{i-2}w_{i-1}w_i$	

```
\data\
ngram 1=1447
ngram 2=9420
ngram 3=5201

\1-grams:
-0.8679678    </s>
-99           <s>                          -1.068532
-4.743076     chow-fun                     -0.1943932
-4.266155     fries                        -0.5432462
-3.175167     thursday                     -0.7510199
-1.776296     want                         -1.04292
...

\2-grams:
-0.6077676    <s>    i                     -0.6257131
-0.4861297    i      want                  0.0425899
-2.832415     to     drink                 -0.06423882
-0.5469525    to     eat                   -0.008193135
-0.09403705   today  </s>
...

\3-grams:
-2.579416     <s>    i          prefer
-1.148009     <s>    about      fifteen
-0.4120701    to     go         to
-0.3735807    me     a          list
-0.260361     at     jupiter    </s>
-0.260361     a      malaysian  restaurant
...
\end\
```

Language Models

- Language models assign a probability that a sentence is a legal string in a language.
- They are useful as a component of many NLP systems, such as ASR, OCR, and MT.
- Simple N-gram models are easy to train on unsupervised corpora and can provide useful estimates of sentence likelihood.
- MLE gives inaccurate parameters for models trained on sparse data.
- Smoothing techniques adjust parameter estimates to account for unseen (but not impossible) events.

LM Tutorial

Homework

- Write two programs
  - train-unigram: Creates a unigram model
  - test-unigram: Reads a unigram model and calculates entropy and coverage for the test set
- Get data from https://github.com/neubig/nlptutorial/tree/master/test
- Test them test/01-train-input.txt test/01-test-input.txt
- Train the model on data/wiki-en-train.word
- Calculate entropy and coverage on data/wiki-entest.word
- Report your scores next week

Pseudo code: train-unigram

```
counts  = {}
total_count = 0
for line in the training_file:
    words = line.split()
    words.append("</s>")
    for word in words:
        counts[word] += 1
        total_count += 1
open the model_file for writing
for word, count in counts:
    probability = counts[word]/total_count
    print word, probability to model_file
```

Pseudo-code: test-unigram

Load model

```
probabilities = {}
for line in model_file:
    w, P = line.split()
    probabilities[w] = P
```

Test and print

```
W = 0
unk = 0
H = 0
for line in test_file:
    words = line.split()
    words.append("</s>")
    for w in words:
        W += 1
        P = λ_unk / V
        if probabilities[w] exists
            P += λ_1 * probabilities[w]
        else
            unk += 1
            H += -log_2(P)
print "entropy = " + H/W
print "coverage = " + (W - unk)/W
```

Summary

- Language Modeling (N-grams)
  - N-grams
  - The Chain Rule
  - The Shannon Visualization Method
- Evaluation:
  - Perplexity
- Smoothing:
  - Laplace (Add-1)
  - Add-k
  - Add-prior