CSEN1076: NATURAL LANGUAGE PROCESSING AND INFORMATION RETRIEVAL

LECTURE 5 – LANGUAGE MODELING

MERVAT ABUELKHEIR

PROBABILISTIC LANGUAGE MODELS

Today's goal: assign a probability to a sentence

- Machine Translation
  - P(**high** winds tonight) > P(**large** winds tonight)
- 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?

他 向 记者 介绍了 主要 内容

he introduced reporters to the main contents of the statement
he briefed to reporters the main contents of the statement
he briefed reporters on the main contents of the statement

PROBABILISTIC LANGUAGE MODELS

Goal: compute the probability of a sentence or sequence of words:

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

Related task: probability of an upcoming word:

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

A model that computes either of these:

$P(W)$ or $P(w_n|w_1 w_2 \ldots w_{n-1})$ is called a **language model**

Better: **the grammar** But **language model** or **LM** is standard

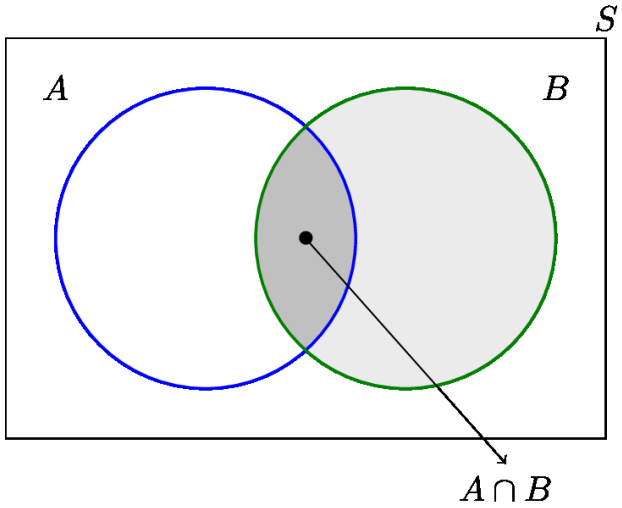
HOW TO COMPUTE $P(W)$

How to compute this joint probability:

- P(its, water, is, so, transparent, that)

Intuition: let's rely on the **Chain Rule of Probability**

THE CHAIN RULE



Recall the definition of conditional probabilities

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

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

More variables:

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

The Chain Rule in General

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

Sequences of N-Grams!

THE CHAIN RULE APPLIED TO COMPUTE JOINT PROBABILITY OF WORDS IN SENTENCE

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i | w_1 w_2 \dots w_{i-1})$$

$$P(\text{“}The\ German\ University\ in\ Cairo\text{”})$$

$$= P(The) \times P(German|The) \times P(University|The\ German)$$

$$\times \ P(in|The\ German\ University)$$

$$\times \ P(Cairo|The\ German\ University\ in)$$

HOW TO ESTIMATE THESE PROBABILITIES

Could we just count and divide?

$$P(Cairo|The\ German\ University\ in) = \frac{count(The\ German\ University\ in\ Cairo)}{count(The\ German\ University\ in)}$$

Relative Frequency: divide observed frequency of a particular sequence by observed frequency of a prefix

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

We'll never see enough data for estimating these counts

Then, you can try and divide by count of all possible 5-word sequences?

- No! Too many possible sentences!

MARKOV ASSUMPTION


Andrei Markov

Simplifying assumption:

$$P(Cairo|German\ University\ in) \approx P(Cairo|in)$$

Or maybe

$$P(Cairo|German\ University\ in) \approx P(Cairo|University\ in)$$

MARKOV ASSUMPTION

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

In other words, we approximate each component in the product by going back in history ***k*** steps [k can be 0, 1, 2, …]

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-k} \dots w_{i-1})$$

SIMPLEST CASE: UNIGRAM MODEL

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

$$P(German\ University\ in\ Cairo) \approx P(German) \times$$

$$P(University) \times P(in) \times P(Cairo)$$

BIGRAM MODEL

Condition on the previous word:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i | w_{i-1})$$

$$P(German\ University\ in\ Cairo) \approx P(German|) \times$$

$$P(University|German) \times P(in|University) \times$$

$$P(Cairo|in)) \times P(|Cairo)$$

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 (MLE)

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Relative Frequency (again):
divide observed frequency of a particular sequence by observed frequency of a prefix

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

AN EXAMPLE

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

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

Language model consists of probabilities:

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

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

MORE EXAMPLES: BERKELEY RESTAURANT PROJECT SENTENCES

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

…

The corpus consists of 9222 sentences with 1446 unique words

RAW BIGRAM COUNTS

Counts for 8 words (out of 1446 words in vocabulary):

	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

Unigram counts, to be used for normalization:

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

Probabilities:

	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

WHAT KINDS OF KNOWLEDGE?

Given the following probabilities (also computed from corpus):

$$P\ (I\ |\ <s>)\ = 0.25,\ P\ (</s>\ |\ food)\ = 0.68$$

$$P(english|want)\ = 0.0011,\ \ P(food|english)\ = 0.5$$

We can compute:

$$P(<s>\ \ I\ want\ english\ food\ \ </s>)$$

$$=\ P(I|<s>) \times P(want|I) \times P(english|want) \times P(food|english) \times P(</s>|food)$$

$$=\ \ .000031$$

PRACTICAL ISSUES

We will do everything in log space …

- to avoid underflow
- (also adding is faster than multiplying)

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

LANGUAGE MODELING TOOLKITS

SRILM

- http://www.speech.sri.com/projects/srilm/

CMU-Cambridge Statistical Language Modeling Toolkit

- http://www.speech.cs.cmu.edu/SLM/toolkit.html

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: HOW GOOD IS OUR MODEL?

Does our language model prefer good sentences to bad ones?

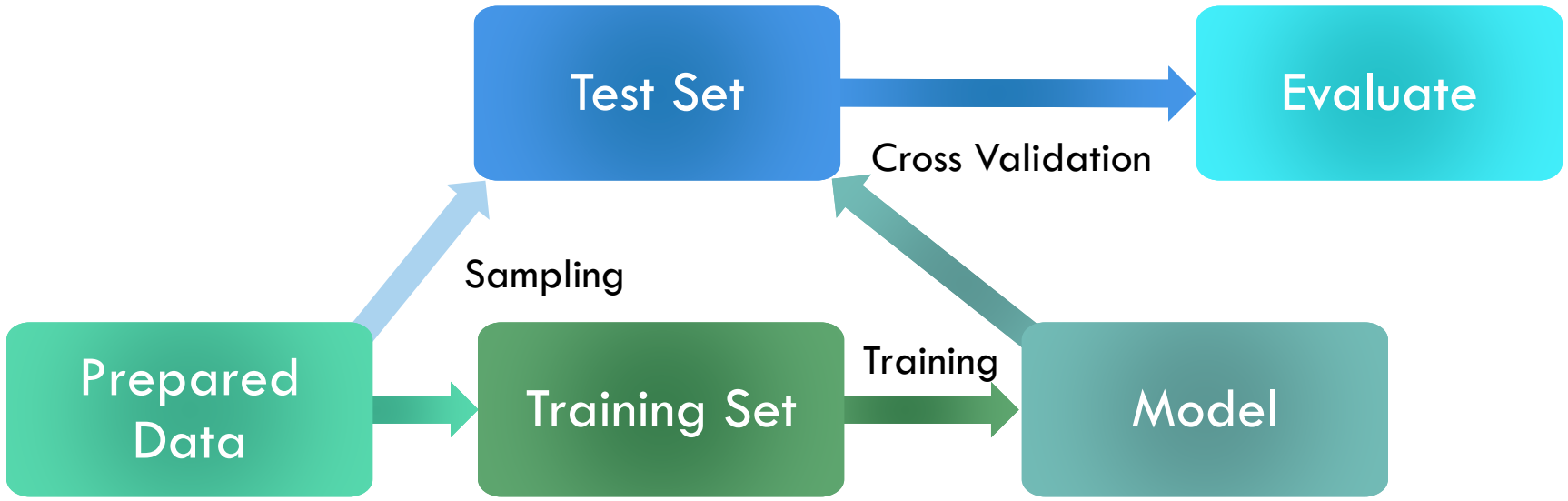
- Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?
  - e.g. a good model should give the sentence “The apple is red” a much higher probability than the sentence “red is apple the”

We train parameters of our model on a **training set**

We test the model’s performance on data we haven’t seen

- A **test set** is an **unseen dataset** that is different from our training set
- An **evaluation metric** tells us how well our model does on the test set

EVALUATION: THE PIPELINE



EXTRINSIC EVALUATION (IN-VIVO) OF N-GRAM MODELS

Best evaluation for comparing models A and B

1. Put each model in a task
   - e.g. spelling corrector, speech recognizer, MT system
2. Run the task, get accuracy for A and for B
   - e.g. how many misspelled words corrected properly
   - e.g. how many words translated correctly
3. Compare accuracy for A and B

DIFFICULTY OF EXTRINSIC EVALUATION OF N-GRAM MODELS

Extrinsic evaluation is time-consuming; can take days or weeks

So

- Sometimes use **intrinsic evaluation: perplexity**
  - In information theory, perplexity is **a measurement of how well a probability model predicts a test sample**
  - Perplexity test is the common LM evaluation metric
  - the test data should look **just** like the training data – but NOT be part of training data!
  - So **generally only useful in pilot experiments**

INTUITION OF PERPLEXITY

The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

- Unigrams are terrible at this game (Why?)

A better model of a text is one which assigns a higher probability to the word that actually occurs

INTUITION OF 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 N:

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

geometric mean normalization

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

Chain rule:

$$PP(W) = \sqrt[N]{\prod_i \frac{1}{P(w_i | w_1 w_2 \ldots w_{i-1})}}$$

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

Model is less surprised by the test sample

PERPLEXITY AS BRANCHING FACTOR

Scenario: a sentence consisting of random digits

What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

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

Perplexity of 10 means that the average branching factor is 10. i.e. the next word could be one out of 10 possible words

LOWER PERPLEXITY = BETTER MODEL

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

RANDOM TEXT GENERATION

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>
I want to eat Chinese food
```

APPROXIMATING SHAKESPEARE

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

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 worse: What's coming out looks like Shakespeare because it is Shakespeare

THE WALL STREET JOURNAL IS NOT SHAKESPEARE

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	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
3 gram	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

THE WALL STREET JOURNAL IS NOT SHAKESPEARE

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 that generalize!
- One kind of generalization: **Smooth the Zeros!**
  - Things that don't ever occur in the training set
  - But occur in the test set

ZEROS

Training set:

… denied the allegations
… denied the reports
… denied the claims
… denied the request

Test set

… denied the offer
… denied the loan

P(“offer” | denied the) = 0

ZERO PROBABILITY BIGRAMS

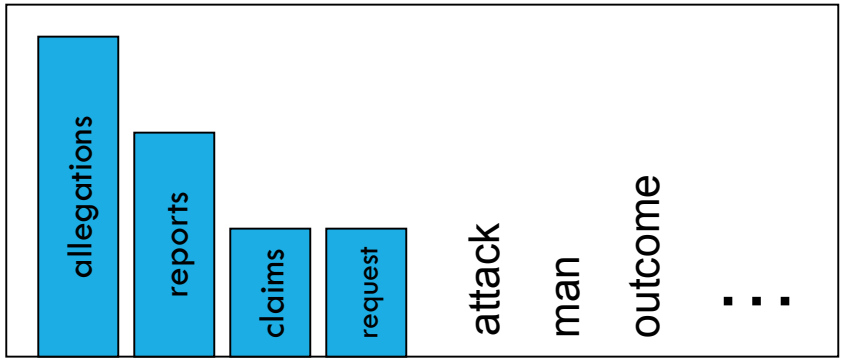
Bigrams with zero probability mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

THE INTUITION OF SMOOTHING

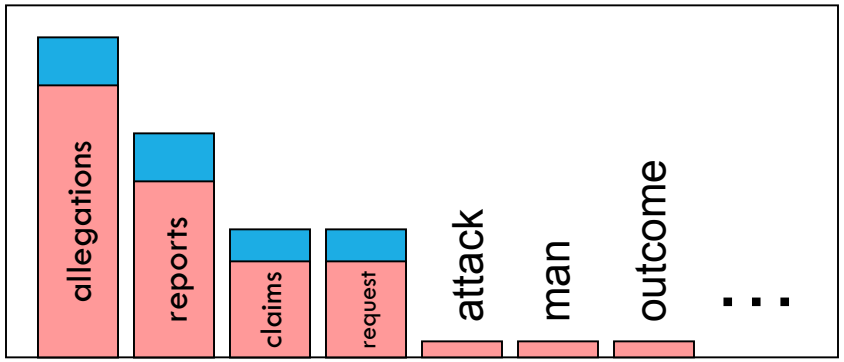
When we have sparse statistics:

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



ADD-1 ESTIMATION (LAPLACE SMOOTHING)

Pretend we saw each word one more time than we did

Just add one to all the counts!

MLE estimate: $$P_{MLE}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Add-1 estimate: $$P_{Add-1}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |V|}$$

MAXIMUM LIKELIHOOD ESTIMATES

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word “bagel” occurs 400 times in a corpus of a million words

What is the probability that a random word from some other text will be “bagel”?

MLE estimate is 400/1,000,000 = 0.0004

This may be a bad estimate for some other corpus

- **But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus**

BERKELEY RESTAURANT CORPUS: LAPLACE SMOOTHED BIGRAM COUNTS

	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

$$P_{Add-1}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |V|}$$

	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

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |V|}$$

Probability mass must stay the same!

RECONSTITUTED COUNTS

$$c(w_{i-1}, w_i) = \frac{[count(w_{i-1}, w_i) + 1] \times count(w_{i-1})}{count(w_{i-1}) + |V|}$$

	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

COMPARE WITH RAW BIGRAM COUNTS

	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

	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

Smoothing is also called **discounting** as the n-gram counts are decreased

ADD-1 ESTIMATION IS A BLUNT INSTRUMENT

So add-1 isn't used for N-grams:

- We'll see better methods

But add-1 is used to smooth other NLP models

- For text classification
- In domains where the number of zeros isn't so huge

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

Interpolation works better

LINEAR INTERPOLATION

Simple interpolation

$$\hat{P}(w_i|w_{i-1}w_{i-2}) = \lambda_1 P(w_i|w_{i-1}w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)$$

$$\sum \lambda_i = 1$$

Lambdas conditional on context:

$$\hat{P}(w_i|w_{i-1}w_{i-2}) = \lambda_1\left(w_{i-2}^{i-1}\right)P(w_i|w_{i-1}w_{i-2}) + \lambda_2\left(w_{i-2}^{i-1}\right)P(w_i|w_{i-1}) + \lambda_3\left(w_{i-2}^{i-1}\right)P(w_i)$$

LINEAR INTERPOLATION

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:

HUGE WEB-SCALE N-GRAMS

How to deal with, e.g., Google N-gram corpus

Pruning

- Only store N-grams with count > threshold
  - Remove singletons of higher-order n-grams
- Entropy-based pruning

Efficiency

- Efficient data structures like tries
- Store words as indexes, not strings
  - Use Huffman coding to fit large numbers of words into two bytes
- Quantize probabilities (4-8 bits instead of 8-byte float)

SMOOTHING FOR WEB-SCALE N-GRAMS

"Stupid backoff" (Brants *et al*. 2007)

No discounting, just use relative frequencies

$$S\left(w_i|w_{i-k+1}^{i-1}\right) = \begin{cases} \dfrac{count\left(w_{i-k+1}^{i}\right)}{count\left(w_{i-k+1}^{i-1}\right)} & if\ count\left(w_{i-k+1}^{i}\right) > 0 \\ 0.4S\left(w_i|w_{i-k+2}^{i-1}\right) & otherwise \end{cases}$$

$$S(w_i) = \frac{count(w_i)}{N}$$

ADVANCED SMOOTHING ALGORITHMS

Many smoothing algorithms other than Laplace:

- Good-Turing
- Kneser-Ney
- Witten-Bell

Intuition used:

Use the count of things we've seen once

- to help estimate the count of things we've never seen

ABSOLUTE DISCOUNTING INTERPOLATION

$$P_{Absolute\_discount}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)-d}{count(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

discounted bigram

unigram

Interpolation weight

- Most used $d = 0.75$

But should we really just use the regular unigram P(w)?

KNESER-NEY SMOOTHING I

$$P_{Absolute\_discount}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)-d}{count(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

discounted bigram

unigram

Interpolation weight

- Most used $d = 0.75$

But should we really just use the regular unigram P(w)?

- Shannon game: *I can't see without my reading*_____?
- "Francisco" is more common than "glasses"
- … but "Francisco" always follows "San"

KNESER-NEY SMOOTHING II

The unigram is useful exactly when we haven't seen this bigram!

Instead of P(w): "How likely is w"

$P_{continuation}(w_i)$: "How likely is w to appear as a novel continuation?

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

$$P_{continuation}(w_i) = \frac{|\{(w_{i-1}): c(w_{i-1}, w_i) > 0\}|}{|\{(w_{j-1}, w_j): c(w_{j-1}, w_j) > 0\}|}$$

KNESER-NEY SMOOTHING III

$$P_{Absolute\_discount}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)-d}{count(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)$$

N-GRAM SMOOTHING SUMMARY

Add-1 smoothing:

- OK for text categorization, not for language modeling

The most commonly used method:

- Extended Interpolated Kneser-Ney

For very large N-grams like the Web:

- Stupid backoff

NEXT TIME

Vector Semantics and Word Embeddings

REFERENCES

This lecture is relying on the following courses:

- Connectionist and Statistical Language Processing slides, Frank Keller, Universitat des Saarlandes
- Natural Language Processing Lecture Slides from the Stanford Coursera course by Dan Jurafsky and Christopher Manning