Bahria University
Department of Computer Science

Adv. Natural Language Processing

Lecture 5 & 6

Instructor: Dr. Muhammad Asfand-e-yar

Previous Lecture

Minimum Edit Distance

Today’s Lecture

- Introduction to N – Grams
- Estimating N-Grams Probabilities
- Evaluation and Perplexity
- Generalization and Zeros
- Laplace Smoothing (Add 1)
- Interpolation and Backoff
- Kneser-Ney Smoothing

Language Modeling

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence
Why?

- 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., etc.!!

Probabilistic Language Modeling

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

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

Related task: probability of an upcoming word:

$$P(w_5 \mid w_1, w_2, w_3, w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n \mid w_1, w_2 \dots w_{n-1})$ is called **a language model**.

Computing Probability can be either by complete combination of words

Or combination of last word with its previous words

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

How to compute *P(W)*

How to compute the joint probability:

For example:

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

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

The Chain Rule

Give any of these word sequences, what is the probability of the next word?

Premature optimization is the root of all <u>evil</u> *-Donald Knuth*

A house divided against itself <u>cann't</u> <u>stand</u> *-Abraham Lincoln*

The quick brown fox jumped over the <u>lazy</u> <u>dog</u> *-Wm. Shakespeare*

A friend to all is <u>a</u> <u>friend</u> <u>of</u> <u>none</u> *-Aristotle*

If you were able to complete these word sequences, it was likely from prior knowledge and exposure to the complete sequence.

Not all word sequences are obvious, but for any given word sequence, it should be possible to compute the probability of the next word.

The Chain Rule

N-Grams

Word sequences are given a formal name:

Unigram	A sequence of one word *WebSphere, Mobile, Coffee*
Bigram	A sequence of two words: *cannot stand, Lotus Notes*
Trigram	A sequence of three words: *Lazy yellow dog, friend to none, Rational Software Architect*
4-Gram	A sequence of four words: *Play it again Sam*
5-Gram	A sequence of five words
6-Gram	A sequence of six words (etc.)

The Chain Rule

What is the probability that "Sam" will occur after the trigram "Play it again"?

The word sequence might well be

1. "Play it again Sally",
2. "Play it again Louise“,
3. or "Play it again and again",
4. and so on.

If we want to compute the probability of "Sam" occurring next, how do we do this?

The chain rule of probability: **P(W) = P(w4 | w1, w2, w3)** This can be stated:

P(W)	"A sequence of words"
=	
P(w4 | w1, w2, w3)	"The conditional probability of word w4 given the sequence w1,w2,w3."

The Chain Rule

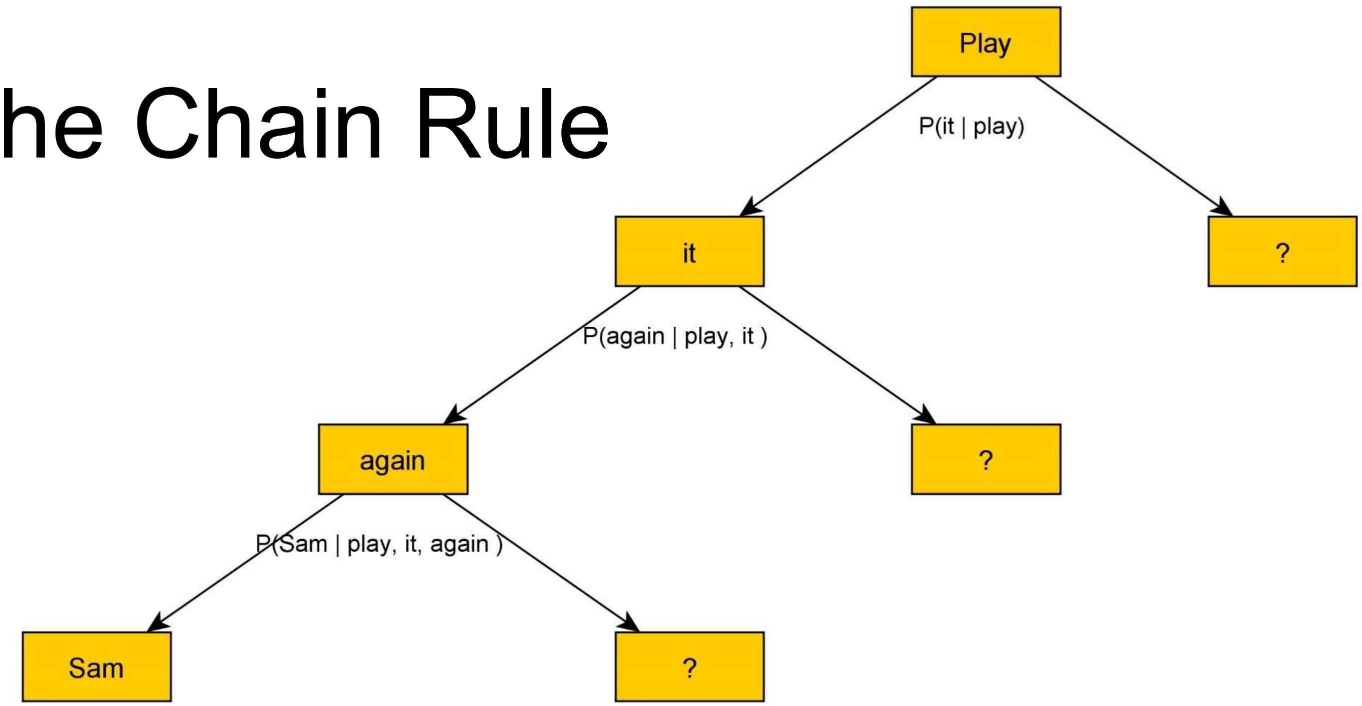
P(W)	"A sequence of words"
=	
P(w4 | w1, w2, w3)	"The conditional probability of word w4 given the sequence w1,w2,w3."

Therefore, if we plug the values for "Play it again Sam" into this formula, we get

P(Sam | Play, it, again)

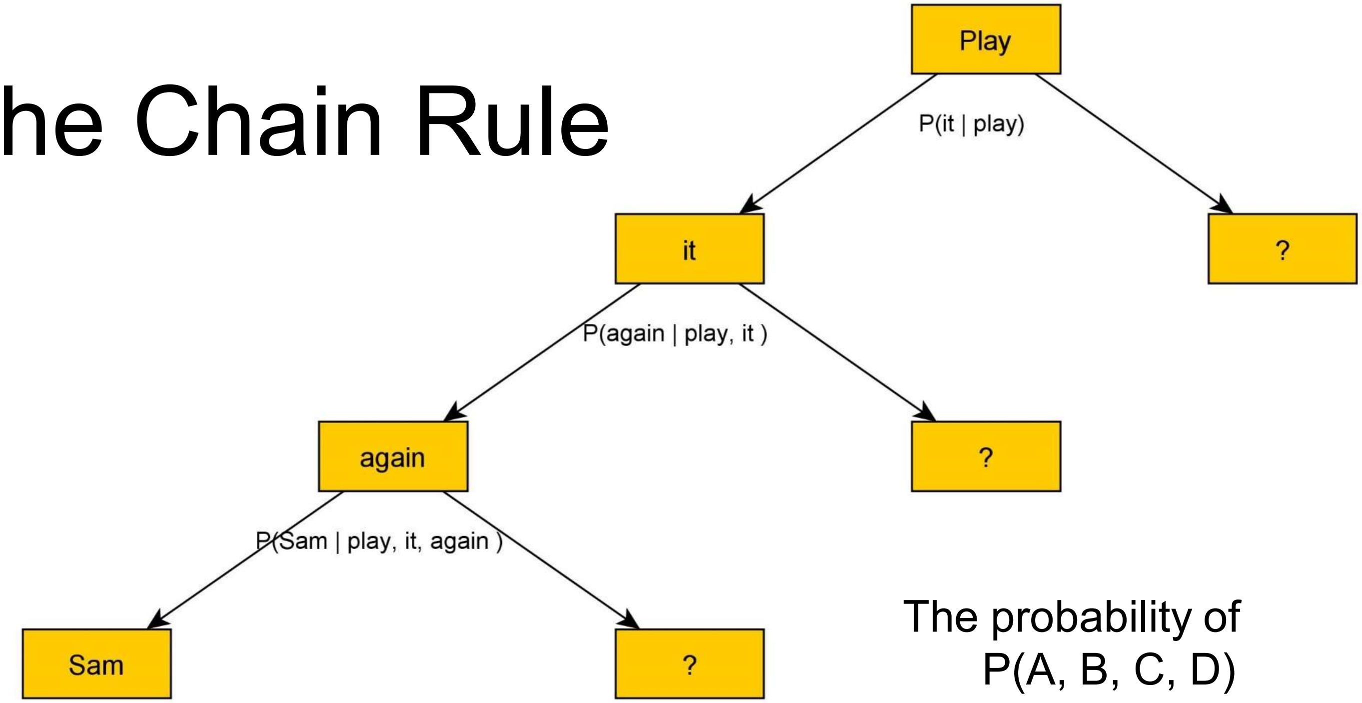
Hence given the word sequence { Play, it, again }, what is the probability of "Sam" being the fourth word in this sequence?
We can answer a question with a question.

The Chain Rule



1. What is the probability that "it" will follow "play"?
2. What is the probability that "again" will follow "play it"?
3. What is the probability that "Sam" will follow "play it again"?

The Chain Rule



The probability of

P(A, B, C, D)

is

P(A) * P(B | A) * P(C | A, B) * P(D | A, B, C)

or with values in place:

P(Play, it, again, Sam)

is

P(Play) * P(it | Play) * P(again | Play, it) * P(Sam | Play, it, again)

The Chain Rule

Recall the definition of conditional probabilities

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

Rewriting: $\mathbf{P(A, B) = P(A) * P(B|A)}$

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

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

The Chain Rule

The Chain Rule applied to compute joint probability of words in sentence

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

P("its water is so transparent") =

P(its) × P(water | its) × P(is | its, water) ×

P(so | its, water, is) × P(transparent | its, water, is, so)

How to estimate these probabilities?

Could we just count and divide?

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

No! Too many possible sentences!

We’ll never see enough data for estimating these
…the *longer* the sequence, the *less likely* we are to find it in a training corpus

Markov Assumption

Simplifying assumption:

Andrei Markov

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

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

In other words, we approximate each component in the product

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

Therefore, according to above equation we can say that the probability of all words approximately equals the last word to its previous few words.

Simplest case: Unigram model

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

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a, a,
the, inflation, most, dollars, quarter, in, is, mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Bigram model

Condition on the previous word:

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

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

N-Gram models

We can extend to tri-grams, 4-grams, 5-grams

In general this is an insufficient model of language

because language has **long-distance dependencies**:

"The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing."

But we can often get away with N-gram models

Language Modeling

Estimating N-gram Probabilities

Estimating Bigram probabilities

How do we estimate these bigram or N-gram probabilities?

The Maximum Likelihood Estimate (MLE)

Get the MLE estimate for the parameters of an N-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.

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

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Estimating Bigram probabilities

Let's work through an example using a mini-corpus of three sentences.

We'll first need to augment each sentence with a special symbol **<s>** at the beginning of the sentence, to give us the bigram context of the first word.

We'll also need a special end-symbol **</s>**.

<s> I am Sam </s>

<s> Sam I am </s>

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

An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

Here are the calculations form some of the bigram probabilities from the above given corpus (i.e. example).

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

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

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

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 counts

Now to Normalize the counts by calculated the Probability unigrams.

It is:

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Raw bigram probabilities

Normalize by unigrams:

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

In the corpus of 9222 sentences the count of each word separately is given in above table.

Raw bigram probabilities

Normalize by unigrams:

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

Normalize by Bigrams:

	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:

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

Raw bigram probabilities

Normalize by unigrams:

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

Note that for probability of 0 will be same in count be 0.

Bigram Estimates of sentence probabilities

Now we can compute the probability of sentences like

I want English food **or** *I want Chinese food*

by simply multiplying the appropriate bigram probabilities together, as follows:

P(i|<s>) = 0.25

P(want| i) = 0.33

P(english|want) = 0.0011

P(food|english) = 0.5

P(</s>|food) = 0.68

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(food | Chinese) = 0.743

P(food | English) = 0.531

P(want | I) = 0.76

P (I | <s>) = .25

I want English food **<u>or</u>** *I want Chinese food*

0.25*0.76*0.0011*0.531 **<u>or</u>** 0.2*0.76*0.0065*0.743

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

Why the
P(english | want) = 0.0011
is less than
P(chinese | want) = 0.0065
It can be just because people like want chinese food more as compared to english food.
Its what the World want (i.e. People want).

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

P(to | want) = 0.66
It’s grammatical because “want” is infinitive and “to” comes after it.

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

P(want | spend) = 0
It's grammatical because "want" and "spend" are verbs and won't come one after the other. It is because grammatical disallowing. Therefore, the "0" here is structural zero.

What kinds of knowledge we get?

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

P(food | to) = 0
It's possible that "to food" never occurred in the sentence and training data. Therefore, this "0" will be called as contagious zero.

Practical Issues

In practice we don't put the probability as probabilities, but in fact it is log probabilities; therefore we do everything in log space/probabilities. There are two reasons to put everything in log probabilities.

1. Avoid underflow
2. (also adding is faster than multiplying)

   therefore multiplying the probabilities, we put a log and add the probabilities to avoid underflow.

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

Language Modeling Toolkits

SRILM

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

KenLM

- https://kheafield.com/code/kenlm/

These are publicly available Language Modeling tools.

Google N-Gram Release, August 2006

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

These are examples from Google 4-grams counts

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 indispensable 40
serve as the individual 234

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

Google Book N-grams

http://ngrams.googlelabs.com/

Another google corpus is also available and you can download large number of corpus according to your requirements.

Language Modeling

Evaluation and Perplexity

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?

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, totally unused.
- An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

Best evaluation for comparing models A and B

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

Therefore, it is called Extrinsic Evaluation, using external evaluation tools to check the models

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

Extrinsic evaluation (also called in-vivo)

Time-consuming; can take days or weeks

Therefore,

- Sometimes use **intrinsic** evaluation: **perplexity**
- Bad approximation
  - unless the test data looks **just** like the training data
  - therefore, **generally only useful in pilot experiments**
- But is helpful to think about.

Perplexity is the bad approximation unless the test data looks just like the training data

Intuition of Perplexity

The Shannon Game (or Shannon's Method):

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 ____

cat, dog, doll, mango, …

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
….
fried rice 0.0001
….
and 1e-100

Unlikely to choose a word

John or J. F. or
J. Canadi or Canadi or …

Therefore, some sentences can be better predictable and some sentences are worst predictable.

Hence, 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

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:

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

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

Chain rule:

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

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

Shannon Game Intuition for Perplexity

The second method of Perplexity is by Josh Goodman=>

How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’?

Perplexity =10

How hard is recognizing (30,000) names at Microsoft?

Perplexity = 30,000

If a system has to recognize

- Operator (occurs 1 time in 4)
- Sales (occurs 1 time in 4)
- Technical Support (occurs 1 time in 4)
- 30,000 names (occurs 1 time in 120,000 each)

Perplexity is 53

Perplexity is weighted equivalent branching factor

The Josh Goodman Perplexity based on the average branching factor

Shannon Game Intuition for Perplexity (example)

Let’s suppose 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?

$$PP(W) = P\,(w_1\, w_2\, w_3 \ldots w_N)^{-\frac{1}{N}}$$

$$PP(W) = P\,\left({}^{1}/_{10}\ {}^{1}/_{10}\ {}^{1}/_{10}\ \ldots\ {}^{1}/_{10}\right)^{-\frac{1}{N}}$$

$$PP(W) = P\,\left({}^{1}/_{10}{}^{N}\right)^{-\frac{1}{N}}$$

$$PP(W) = P\,\left({}^{1}/_{10}\right)^{-1}$$

$$PP(W) = P\ 10$$

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

(Wall Street Journal)

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The Lower the Perplexity the Better the Language Model

Language Modeling

Generalization and zeros

Generalization and zeros

In previous Lecture

We observed the Uni-grams, Bi-grams and probabilities will be zero in some or major cases.

What should be done with the zeros?

The Shannon Visualization Method

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

Therefore, the Shannon Visualization method show us lot of things about N-grams that we built.

Approximating Shakespeare

For example a grammar model trained by the Shakespeare, and generating random sentences.

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 corpus produces words (N=884, 647) from vocabulary of (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

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

Can you guess the author of these random 3-gram sentences?

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 gram Brazil on market conditions

This shall forbid it should be branded, if renown made it empty.

"You are uniformly charming!" cried he, with a smile of associating and now and then I bowed, and they perceived a chaise and four to wish for.

The perils of overfitting

Therefore, the lesson from above discussion is:

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: 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

Corpus test set

P("offer" | denied the) = 0

Not in Corupus test set

Test set

… denied the offer
… denied the loan

If the above is a given test set of the corpus then what will be the output?

The output definitely will be zero.→ bad job

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)!

Therefore, we have to solve the zero probability bigrams.

Language Modeling

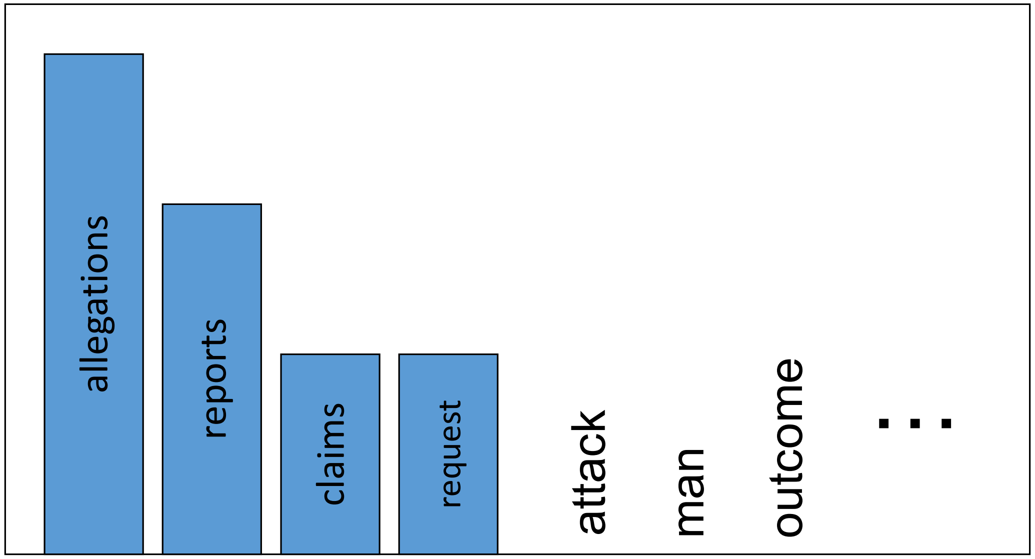
Smoothing: Add-one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

P(w | denied the)

3 allegations
2 reports
1 claims
1 request
7 total

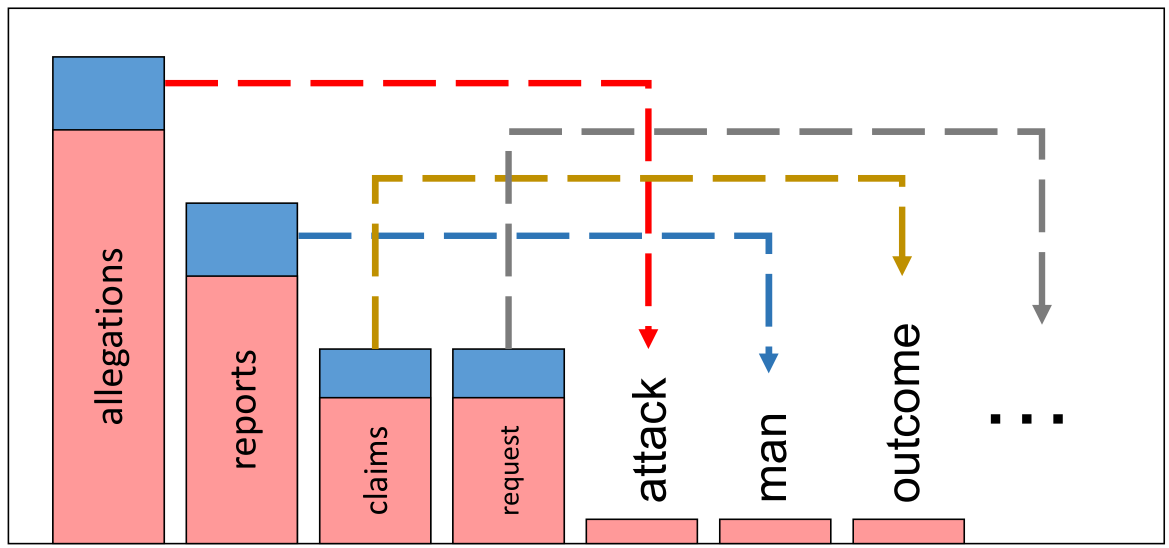


The intuition of smoothing (from Dan Klein)

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



This method is called Add-one estimation or Laplace smoothing

Add-one estimation

Also called Laplace smoothing

Pretend we saw each word one more time than we did

Just add one to all the counts!

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

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

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 = .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.

Therefore, by adding 1 to the MLE is not the likelihood estimate that word occurred in the trained corpus

Berkeley Restaurant Corpus: Laplace Smoothing 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 Smoothing Bigrams

Let's start with the application of Laplace smoothing to unigram probabilities.

Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N:

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

Laplace smoothing merely adds one to each count (hence its alternate name add one smoothing). Since there are V words in the vocabulary and each one was incremented, we also need to adjust the denominator to take into account the extra V observations.

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

Laplace Smoothing Bigrams

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

$$P^*(w_n \mid w_{n-1}) = \frac{c(w_{n-1}\, w_n) + 1}{c(w_{n-1}) + V}$$

Laplace Smoothing Bigrams

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

If we want to calculate the probabilities for following table then V is approximately equal to 1446

V ≈ 1446

	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

Laplace Smoothing Bigrams

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

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

$$P^*(w_i) = \frac{c_i}{N} \quad \left\{ \begin{array}{l} P(w_i) = P(w_n \mid w_{n-1}) \\ c_i = c(w_{n-1}\ \ w_n) \\ N = c(w_{n-1}) \end{array} \right.$$

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

Reconstituted counts

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

	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

Original 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

Reconstituted counts

	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

Add-1 estimation is a blunt instrument

Therefore, 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.

Add-K estimation

Add-k smoothing One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events.

Instead of adding 1 to each count, we add a fractional count k (.5, .05, .01). This algorithm is therefore called add-k smoothing.

$$P^{*}_{\text{Add-k}}(w_{n-1}\ w_n) = \frac{c(w_{n-1}\ w_n) + k}{c(w_{n-1}) + kV}$$

Add-k smoothing requires that we have a method for choosing k; this can be done, for example, by optimizing on a devset (development test set).

Although add-k is useful for some tasks (including text classification), it turns out that it still doesn't work well for language modeling, generating counts with poor variances and often inappropriate discounts (Gale and Church, 1994).