Language Modeling

Applications: Context Sensitive Spelling Correction

The office is about fifteen minuets from my house

min·u·et *noun* \ˌmin-yə-ˈwet\

: a slow, graceful dance that was popular in the 17th and 18th centuries

: the music for a minuet

Use a Language Model

P(about fifteen **minutes** from) > P(about fifteen **minuets** from)

Probablilistic Language Models: Applications

Speech Recognition

- P(I saw a van) $>>$ P(eyes awe of an)

Machine Translation

Which sentence is more plausible in the target language?

- P(**high** winds) > P(**large** winds)

Other Applications

- Context Sensitive Spelling Correction
- Natural Language Generation
- ...

Completion Prediction

- Language model also supports predicting the completion of a sentence.
  - Please turn off your cell ...
  - Your program does not ...
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

Probabilistic Language Modeling

- **Task 1:** Compute the probability of a sentence or sequence of words:

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

- **Task 2:** Probability of an upcoming word:

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

- A model that computes either of these is called a **language model**

The probability of a sentence

How to compute the joint probability

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

Basic Idea

Rely on the Chain Rule of Probability

The Chain Rule

Conditional Probabilities

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

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

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

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

The probability of sentences

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

P("its water is so transparent") =

P(its) x P(water | its) x P(is | its water) x P(so | its water is) x P(transparent | its water is so)

Estimating the probability values

Count and divide

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

What is the problem

We may never see enough data for estimating these

Markov assumption

Simplifying Assumption: Use only the previous word

P(that | its water is so transparent) ≈ P(that | transparent)

Or the couple previous words

P(that | its water is so transparent) ≈ P(that | so transparent)

Markov assumption

More Formally: kth order Markov Model

Chain Rule:

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

Using Markov Assumption: only k previous words

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

We approximate each component in the product

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

N-Gram Models

P(that | its water is so transparent)

An N-gram model uses only $N-1$ words of prior context.

- Unigram: P(that)
- Bigram: P(that | transparent)
- Trigram: P(that | so transparent)

Markov model and Language Model

An N-gram model is an $N-1$-order Markov Model

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- In general, an insufficient model of language:

  language has long-distance dependencies:
  “The computer which I had just put into the machine room on the fifth floor **crashed**.”
- In most of the applications, we can get away with N-gram models

Estimating N-grams probabilities

Maximum Likelihood Estimate

Value that makes the observed data the "most probable"

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

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

An Example

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

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

Estimating bigrams

An Example

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

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

Estimating bigrams

P(I|<s>) =
P(Sam|<s>) =
P(am|I) =
P(</s>|Sam) =
P(Sam|am) =
P(do|I) =

An Example

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

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

Estimating bigrams

P(I|<s>) = 2/3
P(Sam|<s>) = 1/3
P(am|I) = 2/3
P(</s>|Sam) = 1/2
P(Sam|am) = 1/2
P(do|I) = 1/3

Bigram counts from 9332 Restaurant 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

https://github.com/wooters/berp-trans

Computing bigram probabilities

Normlize by unigrams

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

Bigram 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

Computing Sentence Probabilities

P(I want chinese food)

= P(I) x P(want | I) x P(chinese | want) x P(food | chinese)

What knowledge does n-gram represent?

- 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

Consider the following three sentences:

Ram read a Novel

Raj read a Journal

Rai read a book

What is the bigram probability of the sentence **"Ram read a book"**
Include start and end symbols in your calculations.

Solution: 0.1111

Practical Issues

Everything in log space

- Avoids underflow
- Adding is faster than multiplying

$$log(p_1 \times p_2 \times p_3 \times p_4) = logp_1 + logp_2 + logp_3 + logp_4$$

Handling zeros

Use smoothing

Google N-grams

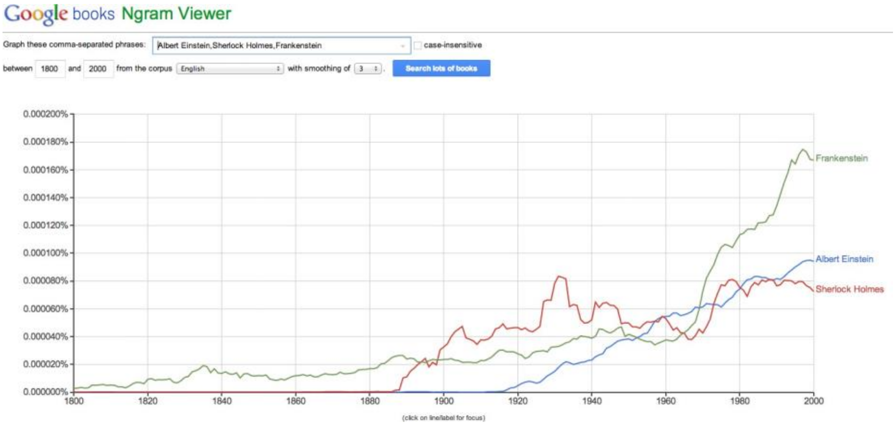
Number of tokens: 1,024,908,267,229
Number of sentences: 95,119,665,584
Number of unigrams: 13,588,391 Number of bigrams: 314,843,401 Number of trigrams: 977,069,902 Number of fourgrams: 1,313,818,354 Number of fivegrams: 1,176,470,663

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

Example from the 4-gram data

serve as the inspector 66
serve as the inspiration 1390
serve as the installation 136
serve as the institute 187
serve as the institution 279
serve as the institutional 461

Google books Ngram Data



Evaluating Language Model

Does it prefer good sentences to bad sentences?

Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Training and Test Corpora

- Parameters of the model are trained on a large corpus of text, called **training set**.
- Performance is tested on a disjoint (held-out) **test data** using an **evaluation metric**

Extrinsic evaluation of N-grams models

Comparison of two models, A and B

- Use each model for one or more tasks: *spelling corrector, speech recognizer, machine translation*
- Get accuracy values for A and B
- Compare accuracy for A and B

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ***. . .***
- The president of India is ***. . .***
- I wrote a ***. . .***

Unigram model doesn't work for this game.

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Unigram model doesn't work for this game.

A better model of text

is one which assigns a higher probability to the actual word

Perplexity

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

Perplexity ($PP(W)$)

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

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

Applying chain Rule

$$PP(W) = \left(\prod \frac{1}{P(w_i|w_1 \ldots w_{i-1})} \right)^{1/N}$$

For bigrams

$$PP(W) = \left(\prod \frac{1}{P(w_i|w_{i-1})} \right)^{1/N}$$

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

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

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

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

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unigram perplexity: 962?

The model is as confused on test data as if it had to choose uniformly and independently among 962 possibilities for each word.

Model evaluation

m_1 → abc, abd, abe, … $\Sigma 1$ 10^6 100

- Measuring accuracy can be inconvenient
  - Say that the model predicts the probability of P("abc"), P("abd"), P("abe"), etc.
  - The sum of all these probabilities must be 1 (it is a multi-class classification problem)
  - For a trigram model, if there are 1 million trigrams, the probability of each trigram (on average) is a small number 1/one million
  - If we have a 4-gram model, these probability values are very small numbers, and floating-point underflow can become as issue

0.0001
0.00003
<<

- Alternately, we can describe the probability of a sequence using perplexity
  - $\text{Perplexity}(c_{1:N}) = P(c_{1:N})^{-1/N}$ $(0.0001)^{\frac{-1}{3}}$
  - Perplexity can be thought of as the reciprocal of probability, normalized by sequence length

Perplexity of character 1 to N

Perplexity (example 1)

- Suppose there are 100 characters in a language L, and our unigram model says they are all equally likely, i.e. P("A") = 1/100, P("B") = 1/100, etc.
- The perplexity of of the model will be 100 for a sequence of any length
  - For a random sequence of length 1
    - Perplexity = $P(\text{"A"})^{-1/1} = 0.01^{-1} = 100$
  - For a random sequence of length 2
    - Perplexity = $P(\text{"AB"})^{-½} = (0.01 * 0.01)^{-½} = (0.01 * 0.01)^{-½} = 0.0001^{-0.5} = 100$
- If some characters were more likely than others, than the model will reflect it, with a perplexity less than 100.

Perplexity (example 2)

Suppose there are 3 characters in a language L, and we have built a unigram model. The probabilities for the 3 characters are given by the models are P("A") = 0.25, P("B") = 0.50, and P("C") = 0.25. What will be the perplexity of for the sequences "AAA" and "ABC"? How will the perplexity change for the two sequences if the probabilities were equal for the 3 characters?

- Perplexity("AAA") = P("AAA")$^{-1/3}$ = $(0.25 * 0.25 * 0.25)^{-1/3}$ = 3.94
- Perplexity("ABC") = P("ABC")$^{-1/3}$ = $(0.25 * 0.50 * 0.25)^{-1/3}$ = 3.13 (less perplex = high probability)

If the probabilities were equal:

- Perplexity("AAA") = P("AAA")$^{-1/3}$ = $(0.33 * 0.33 * 0.33)^{-1/3}$ = 2.99
- Perplexity("ABA") = P("ABC")$^{-1/3}$ = $(0.33 * 0.33 * 0.33)^{-1/3}$ = 2.99

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability
- Choose a random bigram (w,x) as per its probability
- And so on until we choose </s>

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

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Zero probability n-grams

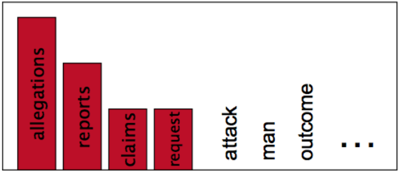
- P(offer | denied the) = 0
- The test set will be assigned a probability 0
- And the perplexity can't be computed

Language Modeling: Smoothing

Language Modeling: Smoothing

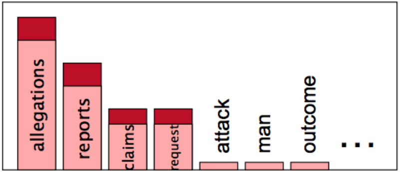
With sparse statistics

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 (Add-one smoothing)

Just add one to all the counts!

MLE estimate for unigram

$$P_{MLE}(w_i) = \frac{c(w_i)}{N} \quad (1)$$

After add-1 smoothing:

$$P_{Add-1}(w_i) = \frac{c(w_i) + 1}{N + V} \quad (2)$$

Laplace Smoothing (Add-one smoothing)

Just add one to all the counts!

MLE estimate for unigram

$$P_{MLE}(w_i) = \frac{c(w_i)}{N} \tag{1}$$

After add-1 smoothing:

$$P_{Add-1}(w_i) = \frac{c(w_i)+1}{N+V} \tag{2}$$

MLE estimate for bigram

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})} \tag{3}$$

After add-1 smoothing:

$$P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)+1}{c(w_{i-1})+V} \tag{4}$$

Reconstituted counts as effect of smoothing

Effective bigram count ($c^*(w_{n-1}w_n)$)

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

Comparing with bigrams: Restaurant corpus

	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

Comparing with bigrams: Restaurant corpus

	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

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

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

Unigram prior smoothing:

$$P_{UnigramPrior}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

A good value of k or m?

Can be optimized on held-out set

Given a corpus C_2, the Maximum Likelihood Estimation (MLE) for the bigram "dried berries" is 0.3 and the count of occurrence of the word "dried" is 580. for the same corpus C_2 the likelihood of "dried berries" after applying add-one smoothing is 0.04. What is the vocabulary size of C_2?

Solution: 0.3 = c(dried berries)/c(dried)
c(dried berries) = 0.3* 580 = 174

Add – 1 smoothing:
0.04 = (174 + 1)/(580 + v)

Vocabulary v = 3795

Advanced smoothing algorithms

Smoothing algorithms

- Good-Turing
- Kneser-Ney

Good-Turing: Basic Intuition

Use the count of things we have see once

- to help estimate the count of things we have never seen