Natural Language Processing

Lecture 6—9/10/2015

Jim Martin

Today

Language modeling with *N*-grams

- Basic counting
- Probabilistic model
  - Independence assumptions

Word Prediction

- Guess the next word...
  - *So I notice three guys standing on the ???*

What are some of the knowledge sources you used to come up with those predictions?

Word Prediction

- We can formalize this task using what are called *N*-gram models
  - *N*-grams are token sequences of length *N*
  - -Our earlier example contains the following 2-grams (aka bigrams)
  - *(So I), (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)*
- Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.

N-Gram Models

- More formally, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence.
- Or, we can use them to assess the probability of an entire sequence of words.
  - Pretty much the same thing as we'll see...

Applications

- It turns out that being able to assess the probability of a sequence is an extremely useful thing to be able to do.
- As we'll see, it lies at the core of many applications
  - Automatic speech recognition
  - Handwriting and character recognition
  - Spam detection
  - Sentiment analysis
  - Spelling correction
  - Machine translation
  - …

Counting

- Simple counting lies at the core of any probabilistic approach. So let's first take a look at what we're counting.
  - *He stepped out into the hall, was delighted to encounter a water brother.*
    - 13 tokens, 15 if we include "," and "." as separate tokens.
    - Assuming we include the comma and period as tokens, how many bigrams are there?

Counting: Types and Tokens

- How about
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - 18 tokens (again counting punctuation)
- But we might also note that "*the*" is used 3 times, so there are only 16 unique types (as opposed to tokens).
- In going forward, we'll have occasion to focus on counting both types and tokens of both words and *N*-grams.
  - When we're looking at isolated words we'll refer to them as unigrams

Language Modeling

- Now that we know how to count, back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence
  - $P(w_n|w_1,w_2\ldots w_{n-1})$
- We'll call a statistical model that can assess this a *Language Model*

Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts. So to get
  - P(*the* | *its water is so transparent that*)
- By definition that's

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

  We can get each of those from counts in a large corpus.

Very Easy Estimate

- How to estimate?
  - P(the | its water is so transparent that)

P(the | its water is so transparent that) =

Count(its water is so transparent that the)
Count(its water is so transparent that)

Very Easy Estimate

- According to Google those counts are 12000 and 19000 so the conditional probability of interest is...

- P(the | its water is so transparent that) = 0.63

Language Modeling

- Unfortunately, for most sequences and for most text collections we won't get good estimates from this method.
  - What we're likely to get is 0. Or worse 0/0.
- Clearly, we'll have to be a little more clever.
  - Let's first use the chain rule of probability
  - And then apply a particularly useful independence assumption

The Chain Rule

- Recall the definition of conditional probabilities

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

- Rewriting:

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

- For sequences...
  - P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)
- In general
  - $P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n-1})$

The Chain Rule

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

P(its water was so transparent)=

P(its)*
P(water|its)*
P(was|its water)*
P(so|its water was)*
P(transparent|its water was so)

Unfortunately

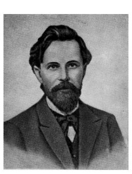
- There are still a lot of possible sequences in there
- In general, we'll never be able to get enough data to compute the statistics for those longer prefixes
  - Same problem we had for the strings themselves

Independence 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)
- That is, the probability in question is to some degree *independent* of its earlier history.

Independence Assumption

- This particular kind of independence assumption is called a *Markov assumption* after the Russian mathematician Andrei Markov.



Markov Assumption

So for each component in the product replace with the approximation (assuming a prefix of N - 1)

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

Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

An Example

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

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

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

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

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*

Bigram Counts

- Vocabulary size is 1446 |V|
- Out of 9222 sentences
  - Eg. "I want" occurred 827 times

	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

Bigram Probabilities

- Divide bigram counts by prefix unigram counts to get bigram probabilities.

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

	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(<s> I want english food </s>) =
  P(i|<s>)*
  P(want|I)*
  P(english|want)*
  P(food|english)*
  P(</s>|food)*
  =.000031

Kinds of Knowledge

- As crude as they are, *N*-gram probabilities capture a range of interesting facts about language.

- 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

World knowledge

Syntax

Discourse

Shannon's Method

- Assigning probabilities to sentences is all well and good, but it's not terribly illuminating. A more entertaining task is to turn the model around and use it to generate random sentences that are *like* the sentences from which the model was derived.
- Generally attributed to Claude Shannon.

Shannon's Method

- Sample a random bigram (<s>, w) according to the probability distribution over bigrams
- Now sample a new random bigram (w, x) according to its probability
  - Where the prefix w matches the suffix of the first bigram chosen.
- And so on until we randomly choose a (y, </s>)
- Then string the words together
- <s> I

  I want

  want to

  to eat

  eat Chinese

  Chinese food

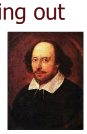
  food </s>

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? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
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 a 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)
  - This is the biggest problem in language modeling; we'll come back to it.
- Quadrigrams are worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare



The Wall Street Journal is Not Shakespeare

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

Model Evaluation

- How do we know if our models are any good?
  - And in particular, how do we know if one model is better than another.
- Well Shannon's game gives us an intuition.
  - The generated texts from the higher order models sure sounds better.
    - That is, they sound more like the text the model was obtained from.
  - The generated texts from the WSJ and Shakespeare models look different
    - That is, they look like they're based on different underlying models.
- But what does that mean? Can we make that notion operational?

Evaluating *N*-Gram Models

- Best evaluation for a language model
  - Put model A into an application
    - For example, a machine translation system
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models
  - ***Extrinsic evaluation***

Evaluation

- Extrinsic evaluation
  - This is really time-consuming and hard
  - Not something you want to do unless you're pretty sure you've got a good solution
- So
  - As a temporary solution, in order to run rapid experiments we evaluate N-grams with an **intrinsic** evaluation
  - An evaluation that tries to capture how good the model is intrinsically, not how much it improves performance in some larger system

Evaluation

- Standard method
  - Train parameters of our model on a **training set**.
  - Evaluate the model on some new data: a **test set**.
    - A dataset which is different than our training set, but is drawn from the same source

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

- Perplexity is the probability of a test set (assigned by the language model), as normalized by the number of words:

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

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

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

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

Lower perplexity means a better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Practical Issues

- Once we start looking at test data, we'll run into words that we haven't seen before. So our models won't wor
- Standard solution
  - Given a corpus
  - Create an unknown word token <UNK>
    Create a fixed lexicon L, of size V
      - From a dictionary or
      - A subset of terms from the training set
    - At text normalization phase, any training word not in L is changed to <UNK>
    - Collect counts for that as for any normal word
  - At test time
    - Use UNK counts for any word not seen in training

Practical Issues

- Multiplying a bunch of really small numbers < 0 is a really bad idea.
  - Multiplication is slow
  - And underflow is likely
- So do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Smoothing

- Back to Shakespeare
  - Recall that 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)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?
  - For generation (shannon game) it means we'll never emit those bigrams
  - But for analysis it's problematic because if we run across a new bigram in the future then we have no choice but to assign it a probability of zero.

Zero Counts

- Some of those zeros are really zeros...
  - Things that really aren't ever going to happen
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count
    - What would that count be in all likelihood?
- Zipf' s Law (long tail phenomenon):
  - 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! We have no counts at all for the vast number of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count) N-grams!

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

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

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 Bigram Probabilities

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

	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

Reconstituted Counts (2)

	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

Big Change to the Counts!

- C(want to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for “chinese food” =.10!!! A 10x reduction
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- Because of this Laplace smoothing not often used for language models, as we have much better methods
- Despite its flaws Laplace (add-1) is still used to smooth other probabilistic models in NLP and IR, especially
  - For pilot studies
  - In document classification
  - In domains where the number of zeros isn’t so huge.

Better Smoothing

- An intuition used by many smoothing algorithms is to use the count of things we’ve seen once to help estimate the count of things we’ve never seen

Types, Tokens and Fish

- Much of what’s coming up was first studied by biologists who are often faced with 2 related problems
  - Determining how many species occupy a particular area (types)
  - And determining how many individuals of a given species are living in a given area (tokens)

One Fish Two Fish

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
    - Not sure where this fishing hole is...
- You have caught up to now
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish to be caught is an eel?
- How likely is it that the next fish caught will be a member of newly seen species?
- Now how likely is it that the next fish caught will be an eel?

Slide adapted from Josh Goodman