AI6122 Text Data Management & Analysis

Topic: N-Gram Model

Outline

- Word prediction
- N-grams
  - Counting and basic concepts
- Language Model
- Evaluation
- Smoothing for LM

Word Prediction

- Guess the next word...
  - Please turn your homework ???
  - I notice three guys standing on the ???

- We can formalize this task using what are called N-gram models.

Word Prediction

- N-grams are token sequences of length N.

- Example sentence: “I notice three guys standing on the”

  – 2-grams (aka bigrams)

    (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)

  – 3-grams (aka trigram)

    (I notice three), (notice three guys), (three guys standing), (guys standing on), (standing on the)

More examples

- Given knowledge of counts of N-grams, we can guess likely next words in a sequence.
  - Predict the next word from the preceding $N-1$ words

- Example sentence: “I notice three guys standing on the ?”
  - From a given dataset (aka corpus, corpora), we have the following counts:
    - “on the street” 50 times,
    - “on the computer” 2000 times,
    - “Standing on the street” 40 times,
    - “standing on the computer” 1 time.

N-Gram Models

- Formally, we use knowledge of the counts of N-grams to assess the conditional probability of candidate words as the next word in a sequence.
  - $P\,(table|I\ notice\ three\ guys\ standing\ on\ the)$
- This is the same as we use them to assess the probability of an entire sequence of words.
  - $P\,(I\ notice\ three\ guys\ standing\ on\ the\ table)$

Applications

- Predict the next word (or any linguistic unit) in a sequence is an extremely useful thing.
  - $P\ (table|I\ notice\ three\ guys\ standing\ on\ the)$
  - $P\ (I\ notice\ three\ guys\ standing\ on\ the\ table)$
- Automatic speech recognition
  - e.g. “I saw a van” vs. “eyes awe of an”
- Handwriting and character recognition
- Spelling correction
  - “Their are problems wit this sentence.”
- Machine translation, and many more.
  - $P(\boldsymbol{high}\ winds\ tonite)\ >\ P(\boldsymbol{large}\ winds\ tonite)$

Counting in Corpora

- Large collections of text

- Brown et al (1992) large corpus of English text
  - 583 million tokens
  - 293,181 types

- Google
  - Crawl of 1,024,908,267,229 English tokens
  - 13,588,391 types (distinct words)
    - That seems like a lot of types...
    - After all, even large dictionaries of English have only around 500k types.
    - Why so many here?

- Numbers
- Misspellings
- Names
- Acronyms
- etc

Language Modeling

- $P(w_1, w_2 \ldots w_{n-1}, w_n)$, the probability of a sequence

- We model the word prediction task as 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 these a Language Model

- How to compute"
  - P(its water is so transparent that the)
  - Let's use the chain rule of probability

The Chain Rule

- Recall the definition of conditional probabilities

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

- Rewriting: $P(A \wedge B) = P(A|B)P(B)$

- For sequences:

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

- In general:
  - $P(x_1, x_2, x_3, \ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \ldots P(x_n|x_1 \ldots x_{n-1})$
  - Notation: we denote $x_1 \ldots x_n$ as x_1^n

The Chain Rule

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

we denote $x_1 \ldots x_n$ as x_1^n

- $P(my\ house\ is\ on\ top\ of\ that\ hill) =$

$$
\begin{aligned}
&P(my) * \\
&\quad P(house|my) * \\
&\quad\quad P(is|my\ house) * \\
&\quad\quad\quad P(on|my\ house\ is) * \\
&\quad\quad\quad\quad P(top|my\ house\ is\ on) * \\
&\quad\quad\quad\quad\quad P(of|my\ house\ is\ on\ top) * \\
&\quad\quad\quad\quad\quad\quad P(that|my\ house\ is\ on\ top\ of) * \\
&\quad\quad\quad\quad\quad\quad\quad P(hill|my\ house\ is\ on\ top\ of\ that)
\end{aligned}
$$

Language Modeling

- How to calculate?
  - $P(hill \mid my\ house\ is\ on\ top\ of\ that)$

- Conditional probabilities: $\frac{P(my\ house\ is\ on\ top\ of\ that\ hill)}{P(my\ house\ is\ on\ top\ of\ that)}$

  - We can get each of those from counts in a large corpus.
  - But not reliable

Independence Assumption

- Make the simplifying assumption
  - Assume: the probability in question is independent of its earlier history.

- If we only consider the previous word, or the previous two words
  - $P(lizard|the, other, day, I, was, walking, along, and, saw, a) = P(lizard|a)$
  - $P(lizard|the, other, day, I, was, walking, along, and, saw, a) = P(lizard|saw, a)$

- This particular kind of independence assumption is called a Markov assumption
  - Bigram version: $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$

Estimating Bigram Probabilities

- The maximum likelihood estimate of $p(w_i \mid w_{i-1})$ from a training set T (corpus)

- The Maximum Likelihood Estimate (MLE)

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

An Example

- Training data (corpus)
  - <s> I am Sam </s>
  - <s> Sam I am </s>
  - <s> I do not like green eggs and ham </s>

- Estimation: “I” will occur after <s> with a probability .67

$$P(\mathrm{I}\,|\,\langle s\rangle) = \frac{2}{3} = .67 \qquad P(\mathrm{Sam}\,|\,\langle s\rangle) = \frac{1}{3} = .33 \qquad P(\mathrm{am}\,|\,\mathrm{I}) = \frac{2}{3} = .67$$

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

Another example dataset

- Berkeley Restaurant Project Sentences

- Example 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

http://www1.icsi.berkeley.edu/Speech/berp.html

Bigram Counts

- 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

- Example: to estimate $P(want \mid i)$
- Divide bigram counts by prefix unigram counts to get 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

Kinds of Knowledge

- N-gram probabilities capture a range of interesting facts about language.
  - $P(english|want) = .0011$
  - $P(chinese|want) = .0065$
  - $P(to|want) = .66$
  - $P(eat \mid to) = .28$
  - $P(food \mid to) = 0$
  - $P(want \mid spend) = 0$

World knowledge

Syntax

Evaluation

- How do we know if our models are any good?

- Standard method
  - Train parameters of our model on a training set.
  - Look at the model performance on some new data (test set)
    - A test set is an unseen dataset that is different from our training set, totally unused (but assumed to be topically similar)

- Then we need an evaluation metric to tell us how well our model is doing on the test set.
  - One such metric is perplexity

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:
  - N : number of words in a test data
  - $w1,..,w_N$ is the test data

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

- Chain rule:

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

- For bigrams:

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

Lower perplexity means a better model

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

- Given a training corpus to build LM, and test data, we can compute perplexity

- $PP(I\ want\ english\ food) = \sqrt[N]{\frac{1}{P(want|I)\times P(english|want)\times P(food|english)}}$

Unknown Words– out of vocabulary (OOV)

- Once we applies the model to test data, we'll run into words that we haven't seen before
  - pretty much regardless of how much training data you have.

- With an Open Vocabulary task
  - Create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - 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 changed to <UNK>
    - Now we count that like a normal word
  - At test time, use UNK counts for any word not in training

Unknown Words– out of vocabulary (OOV)

- Consider the following training corpus

> very good tennis player in US Open
> tennis player US Open
> tennis player qualify play US Open

- Training time
  - Randomly select a subset of vocabulary
    - should not select the high frequency words
    - e.g. 1/3 – {open, qualify, play}
  - Replace the selected words with <UNK>

> very good tennis player in US <UNK>
> tennis player US <UNK>
> tennis player <UNK> <UNK> US <UNK>

Unknown Words– out of vocabulary (OOV)

- Training time, estimate additional probabilities of <UNK>
  - $P(US| < UNK >) = 1/5$
  - $P(< UNK > |US) = 3/3$
  - $P(< UNK > | < UNK >) = 1/5$

- Test time
  - Consider a test sentence “player US Olympic”
  - Replace unseen words with <UNK>: player US <UNK>
  - Estimate the probability of the sentence
    - $P(US|player) \times P(< UNK > |US)$

Zero Counts for Bi-grams

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.

- Some of them are just rare events.
  - If the training corpus had been a little bigger they might (or would) have had a count (probably a count of 1).

Training set:
… denied the allegations
… denied the reports
… denied the claims
… denied the request

Test set
… denied the offer
… denied the loan

$$P(offer \mid denied\ the) = 0$$

Zero Counts

- 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

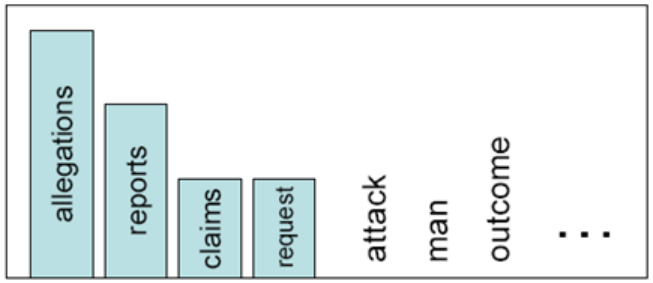
- Result:
  - Our estimates are sparse!
  - We have no counts at all for the vast bulk of things we want to estimate!
  - Estimate the likelihood of unseen (zero count) N-grams!

Insert from others

- Smoothing is like Robin Hood
  - Steal from the rich and give to the poor (in probability mass)

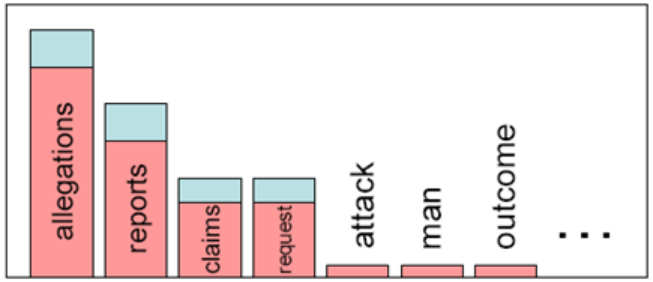
- We often want to make predictions from sparse statistics:

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



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Very important all over NLP, but easy to do badly!

Laplace Smoothing

- Also called add-one smoothing, just add one to all the counts!
- MLE estimate: $$P(w_i) = \frac{c_i}{N}$$
- Laplace estimate:

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

Laplace-Smoothed 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	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	0.002	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

	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: big change to the counts!

- $P(to|want)$ from .66 to .26!
  - Despite its flaws, Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially in domains where the number of zeros isn't so huge.
- There are many other smoothing methods
  - E.g. Backoff and Interpolation.
- If we are estimating trigram $p(z|x,y)$
  - but $count(xyz)$ is zero
  - Use info from bigram $p(z|y)$
  - Or even from unigram $p(z)$
- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff vs. Interpolation

- Backoff:
  - use trigram if you have it, otherwise bigram, otherwise unigram

- Interpolation: mix all three
  - Simple interpolation

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

  - More complicated interpolation available in textbook

Practical Issues

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

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

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

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

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

Google Caveat

- Test sets and training sets should be “similar”
  - drawn from the same distribution for the probabilities to be meaningful.
- The Google corpus is fine if your application deals with arbitrary English text on the Web.
- If not, then a smaller domain specific corpus is likely to yield better results.

Summary and sources

- Word prediction
- N-grams
  - Counting and basic concepts
- Language Model
  - Evaluation
  - Smoothing for LM