CZ4045 Natural Language Processing

N-Grams and Language Models

(Chapter 4)

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 such as these, 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 ?
  - Given a dataset (aka **corpus**, **corpora**),
  - “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 can use knowledge of the counts of N-grams to assess **the conditional probability** of candidate words as the next word in a sequence.
  - $P\ (table|I\ notice\ three\ guys\ standing\ on\ the)$

- Or, we can use them to assess the **probability** of an entire sequence of words.
  - $P\ (I\ notice\ three\ guys\ standing\ on\ the\ table)$

- Pretty much the same thing as we'll see...

Applications

- It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing.

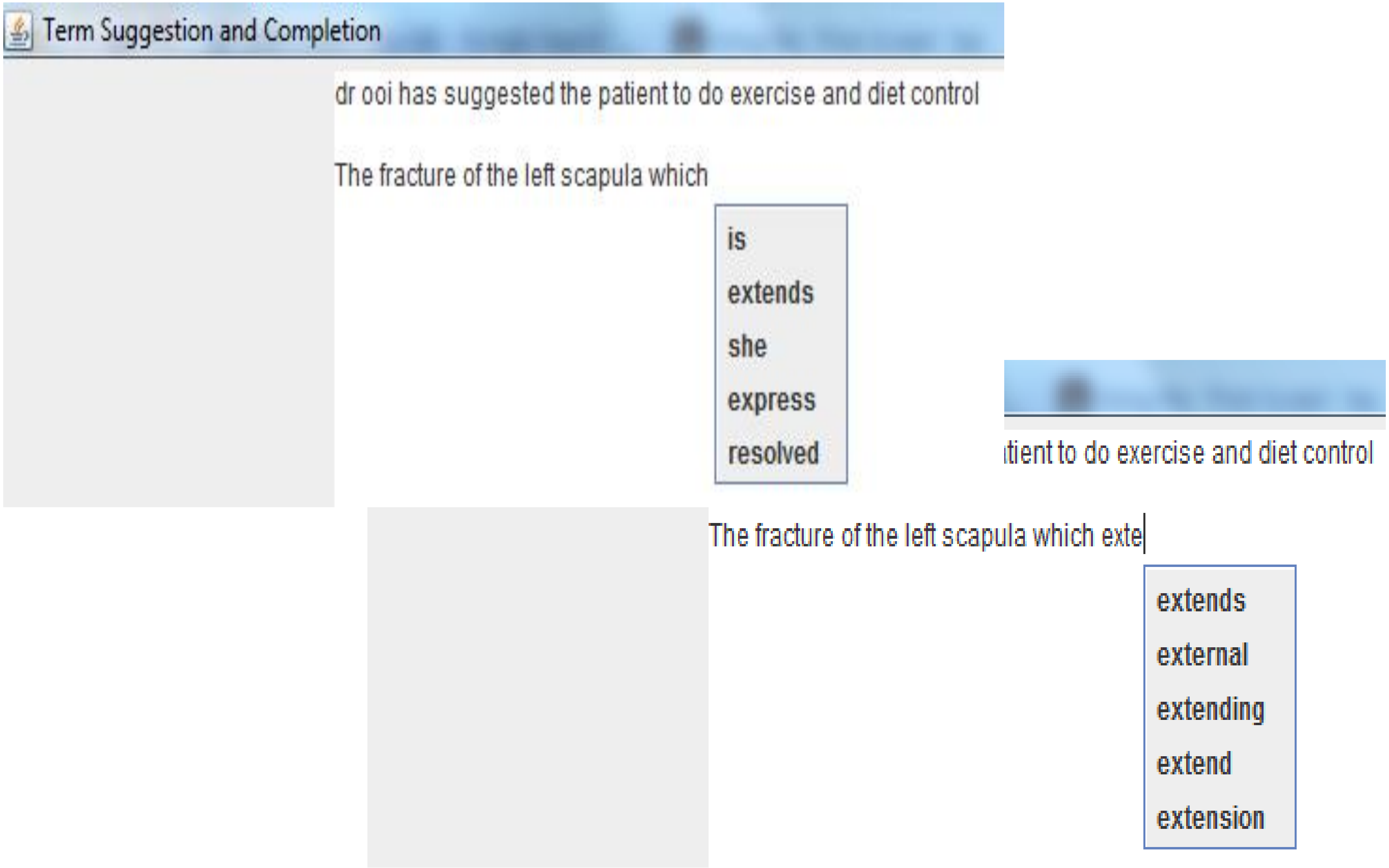
- Automatic speech recognition
  – e.g. “I saw a van” vs. “eyes awe of an”

- Handwriting and character recognition

- Spelling correction
  – “Their are problems wit this sentence.”

- Machine translation, and many more.

Clinical applications



Counting

- Simple counting lies at the core of any probabilistic approach. So let's first take a look at **what we're counting**.

- Example sentence
  - He stepped out into the hall, was delighted to encounter a water brother.

- Counting
  - 13 tokens, 15 if we include "," and "." as separate tokens.
  - Assuming we include the comma and period, how many bigrams are there?

Counting: Types and Tokens

- Example sentence.
  - 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.

Counting: 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
    - 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

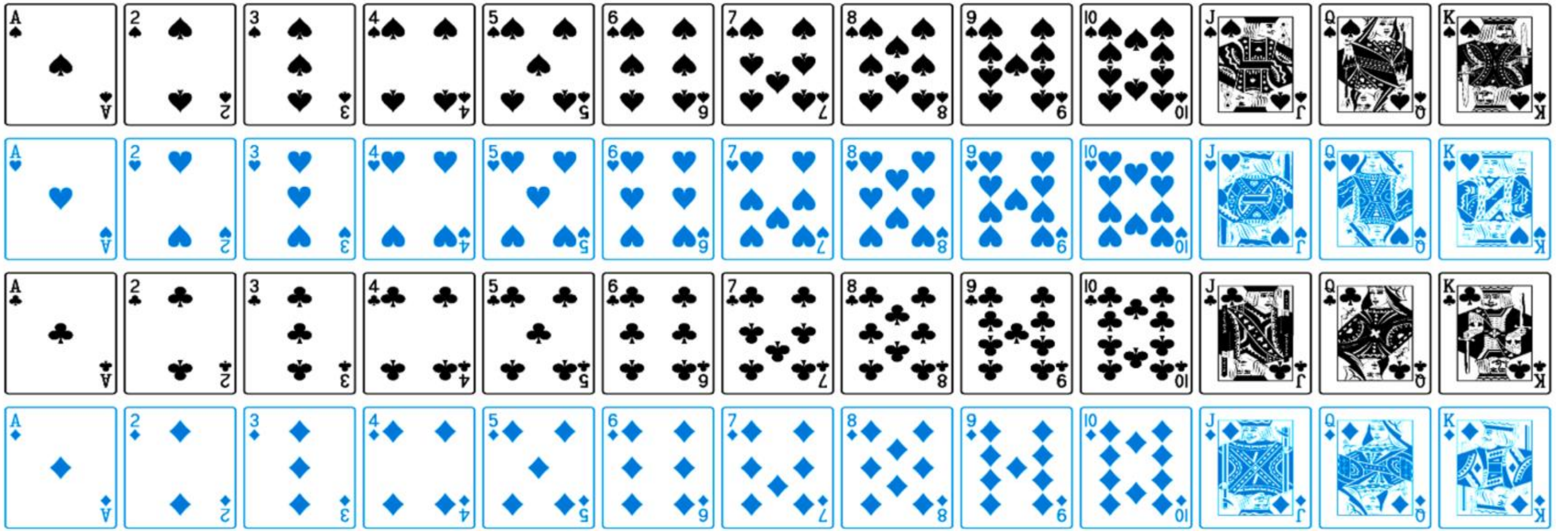
- $P(w_1, w_2 \dots w_{n-1}, w_n)$, the probability of a sequence

- We can model the word prediction task as the conditional probability of a word given the previous words in the sequence
  - $P(w_n | w_1, w_2 \dots w_{n-1})$
  - We'll call a statistical model that can assess these a **Language Model**

- How to compute
    - P(its water is so transparent that the)

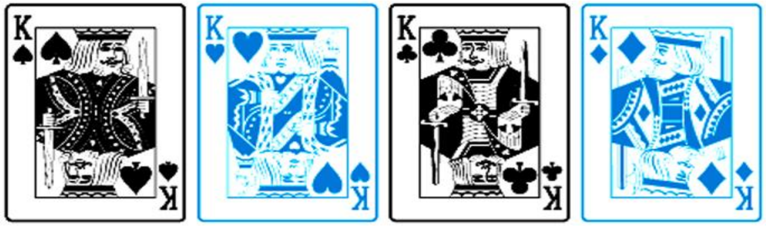
- Let's use the chain rule of probability

Recall a few probability basics--A deck of playing cards



- The event the king of hearts is selected: 1/52

- The event a king is selected: 1/13



The Conditional-Probability Rule

	Rank				
	Full professor R_1	Associate professor R_2	Assistant professor R_3	Instructor R_4	Total
Under 30 A_1	2	3	57	6	68
30–39 A_2	52	170	163	17	402
40–49 A_3	156	125	61	6	348
50–59 A_4	145	68	36	4	253
60 & over A_5	75	15	3	0	93
Total	430	381	320	33	1164

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

$$P(R_3|A_4) = \frac{36}{253} = 0.142$$

$$P(A_4|R_3) = \frac{36}{320} = 0.112$$

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:

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

- Notation: we denote $x_1 \dots x_n$ as x_1^n

The Chain Rule

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

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

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

Language Modeling

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

- By definition of conditional probabilities
  - $\dfrac{P(my\ house\ is\ on\ top\ of\ that\ hill)}{P(my\ house\ is\ on\ top\ of\ that)}$
  - We can get each of those from counts in a large corpus.

Very Easy Estimate

- How to calculate?

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

- By counting:

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

$$= \frac{Count(my\ house\ is\ on\ top\ of\ that\ hill)}{count(my\ house\ is\ on\ top\ of\ that)}$$

Very Easy Estimate

- According to Google those counts are 3 and 7.
  - invalid probability!
  - Even if acceptable (e.g. 3/7), that's not terribly convincing due to the small numbers involved.

- 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.

- Thus, use n-grams (n is small)

Uses of language models

- Text classification
  - sports, finance, politics, etc
  - e.g. $P(Obama\ criticized\ for\ vacation \mid politics)$

- Gender/style detection
- Information retrieval

- Text compression
  - better than gzip for human language text

Summary

- There are still a lot of possible sentences

- In general, we'll never be able to get **enough data** to compute the statistics for **those longer** prefixes
  - $P(hill \mid my\ house\ is\ on\ top\ of\ that)$

- Same problem we had for the strings themselves
  - $P(my\ house\ is\ on\ top\ of\ that\ hill)$

Probabilistic language models

- Three major components
  - Decomposition
    - Sentences are decomposed into n-grams
  - Discounting
    - Save some probability mass for the possibility of unseen events
  - Backoff
    - Alternative choices of n-grams, depending on contexts

Independence Assumption

- Make the simplifying assumption
  - $P(lizard|the, other, day, I, was, walking, along, and, saw, a) = P(lizard|a)$?
- Or maybe
  - $P(lizard|the, other, day, I, was, walking, along, and, saw, a) = P(lizard|saw, a)$?
- Or maybe ??

- That is, the probability in question is independent of its earlier history.

Independence Assumption

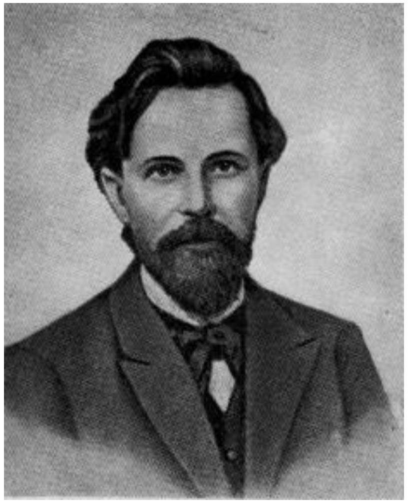
- This particular kind of independence assumption is called a Markov assumption after the Russian mathematician Andrei Markov.

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

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

- Bigram version

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



Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

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

An Example

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

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

- But it is the estimate that makes it most likely that “I” will occur after <s> with a probability .67

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

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

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

- $P(want \mid i)$
- Divide bigram counts by prefix unigram counts to get probabilities.

$$P(want \mid i)$$

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

Exercise: Bigram Maximum Likelihood Estimate

- $P(I\ want\ chinese\ food)\ =$
  - $P(I\ want\ chinese\ food)\ =$

Kinds of Knowledge

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

World knowledge

Syntax

Shannon's Method

Skip

- Assigning probabilities to sentences is all well and good.

- A 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

Skip

- Sample a random bigram (<s>, w) according to its probability
- Now sample a random bigram (w, x) according to its probability
  - Where the prefix w matches the suffix of the first.
- 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

Skip

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

Skip

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

The Wall Street Journal is Not Shakespeare

Skip

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

Questions

Skip

- Well Shannon's game gives us an intuition—higher order models look better

- Quadrigrams can be used for word prediction? Can we make that notion operational?
  - The higher order models are likely to be pretty useless, especially from a small corpus
  - The probability $p(Indeed\ the\ short\ and)$ is very small

Summary

Skip

- Markov Assumption

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

- Maximum Likelihood Estimate (MLE)

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

- Shannon's method for automatic sentence generation
- How good is a model?
- How to deal with unknown words in a new sentence?

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**
  - So use a **test set**.
    - A dataset which is different from our training set
  - Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
    - One such metric is **perplexity** (to be introduced below)

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
  - $w_1, \ldots, w_N$ is the test data

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

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

- For bigrams:

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

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)

- But once we start looking at 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 (Cont'd)
  - 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

- 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?
- Some of those zeros are really zeros...
  - Things that really can't or shouldn't 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
  - (probably a count of 1!).

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

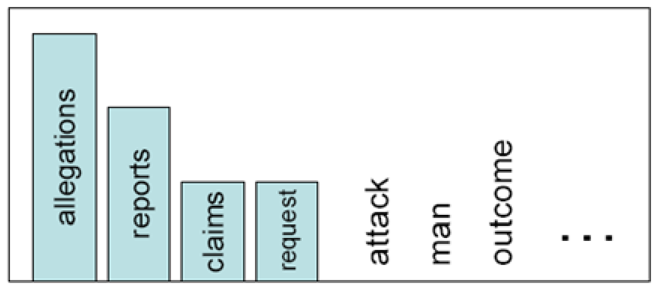
- Result:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count) N-grams!

Insert from others

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

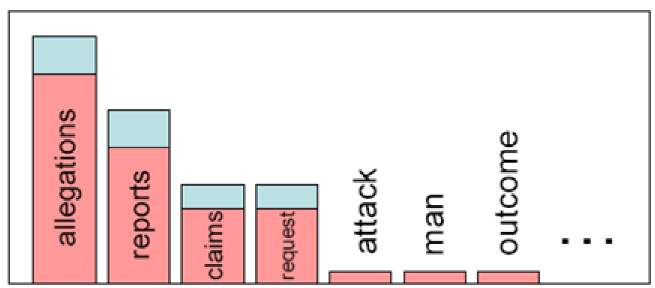
▪ We often want to make predictions from sparse statistics:

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

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.

Better Smoothing

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

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

- (Skip details)

Backoff and Interpolation

- Another really useful source of knowledge

- 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

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

  - More complicated interpolation (read book)

Practical Issues

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

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

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

Language Modeling Toolkits

- SRILM (http://www-speech.sri.com/projects/srilm/)
- CMU-Cambridge LM (http://mi.eng.cam.ac.uk/~prc14/toolkit.html)
- Can use them to get N-gram models
- Lots of parameters (need to know the theory)

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
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.

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

Google Caveat

- Remember the lesson about test sets and training sets...
  - Test sets should be similar to the training set (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
- Great sources for NLP tools!
  - http://nlp.stanford.edu/links/statnlp.html