N - GRAMS

Counting Words in Corpora (1)

Guessing the next word (or **word prediction**) is an essential subtask of speech recognition, hand writing recognition, augmentative communication and spelling error detection.

Augmentative communication systems are computer systems that help the disable in communication.

Detecting real-word spelling error is difficult. There are a number of different machine learning algorithms, which make use of the surrounding words and other features to do **context-sensitive spelling error correction**.

Counting Words in Corpora (2)

An **N-gram** model uses the previous N-1 words to predict the next one.

Such statistical models of word sequences, depending on the context they are being analyzed, are call **language models**, **LM** or **grammar**.

Statistical processing of natural language is based on **corpora** (singular **corpus**) - on-line collections of text and speech. For computing word probabilities, we will count words on training corpus.

Counting Words in Corpora (3)

Utterance – sentence of spoken language.
Fragments – words that are broken in the middle.
Word-form – inflected form as it appears in the corpus.
Lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word-sense.
Types – the number of distinct words in corpus that is the size of the vocabulary.

Tokens – the total number of running words.

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.

APPLICATIONS:

Automatic speech recognition
Handwriting and character recognition
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.

Not always that simple

I do uh main- mainly business data processing

Spoken language poses various challenges.

Should we count "uh" and other fillers as tokens?

What about the repetition of "mainly"?

The answers depend on the application. If we're focusing on something like ASR to support indexing for search, then "uh" isn't helpful (it's not likely to occur as a query). But filled pauses are very useful in dialog management, so we might want them 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.

Counting: Corpora

So what happens when we look at large bodies of text instead of single utterances?

Brown et al (1992) large corpus of English text

- 583 million wordform tokens
- 293,181 wordform types

Google

- Crawl of 1,024,908,267,229 English tokens
- 13,588,391 wordform 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

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

$P(w_n|w_1,w_2...w_{n-1})$

We'll call a statistical model that can assess this a *Language Model*

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

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

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) =

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

According to Google those counts are 5/9.

But 2 of those were to these slides… So maybe it's really 3/7

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

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

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)

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

The **bigram** model approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by the conditional probability of preceding word $P(w_n|w_{n-1})$

This assumption that the probability of a word depends only on the previous word is called **Markov** assumption. Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past.

N-gram looks n-1 words in the past.

A bigram is called a **first-order** Markov model, a trigram is a **second-order** Markov model, and an N-gram is a **N-1th order** Markov model.

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 independent of its earlier history.

Markov Assumption

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

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

Bigram version

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

Estimating Bigram Probabilities

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

Bigram Estimates of Sentence Probabilities

<s> I	0.25
I want	0.32
want to	0.65
to eat	0.26
eat British	0.02
British food	0.60

P(I want to eat British food) = P(i|<s>) P(want|I) P(to|want)P(eat|to)P(British|eat) P(food|British)

= 0.000016

An Example

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

$P(\text{I}|\text{<s>})$

$P(\text{Sam}|\text{<s>})$

$P(\text{am}|\text{I})$

$P(\text{</s>}|\text{Sam})$

$P(\text{Sam}|\text{am})$

$P(\text{do}|\text{I})$

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

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

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

Maximum Likelihood Estimates

The maximum likelihood estimate of some parameter of a model M from a training set T is the estimate that maximizes the likelihood of the training set T given the model M

Suppose the word "*Chinese*" occurs 400 times in a corpus of a million words (Brown corpus)

What is the probability that a random word from some other text from the same distribution will be "Chinese"

MLE estimate is 400/1000000 = .004

This may be a bad estimate for some other corpus

But it is the **estimate** that makes it **most likely** that "Chinese" will occur 400 times in a million word corpus.

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

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 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

N-grams Issues

- Sparse data
  - Not all N-grams found in training data, need smoothing
- Change of domain
  - Train on WSJ, attempt to identify Shakespeare – won't work
- N-grams more reliable than (N-1)-grams
  - But even more sparse
    - Generating Shakespeare sentences with random unigrams…
      - Every enter now severally so, let
    - With bigrams…
      - What means, sir. I confess she? then all sorts, he is trim, captain.
    - Trigrams
      - Sweet prince, Falstaff shall die.

Smoothing Techniques

Every N-gram training matrix is sparse, even for very large corpora (Zipf's law)

Solution: estimate the likelihood of unseen N-grams

Laplace (Add-one) Smoothing

Add 1 to every N-gram count

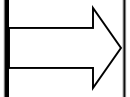
$$P(w_n|w_{n-1}) = C(w_{n-1}w_n)/C(w_{n-1})$$

$$P(w_n|w_{n-1}) = [C(w_{n-1}w_n) + 1] / [C(w_{n-1}) + V]$$

Add-one Smoothed Bigrams

$$P(w_n|w_{n-1}) = C(w_{n-1}w_n)/C(w_{n-1})$$

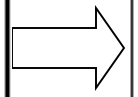
	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0



	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

$$P'(w_n|w_{n-1}) = [C(w_{n-1}w_n)+1]/[C(w_{n-1})+V]$$

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1



	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048