BIGRAMS AND TRIGRAMS

John Fry
Boise State University

n-grams

- We can extend the notion of the probability of a word to the probability of a *pair* of words
- A sequence of two words (e.g., *of the*) is called a *bigram*
- A three-word sequence (e.g., *sound of the*) is called a *trigram*
- The general term n-*gram* means 'sequence of length n'

n	Name	Example
1	unigram	*sky*
2	bigram	*the sky*
3	trigram	*across the sky*
4+	4-gram?	*comes across the sky*

Probability of a bigram $w_1\ w_2$

- $P(w_1\ w_2)$ denotes the probability of the word pair $w_1\ w_2$ ocurring in sequence (e.g., *of the*, *say unto*, etc.)
- If we make one minor simplifying assumption, then the formula is exactly the same as for a single word:

$$P(w_1\ w_2) = \frac{c(w_1\ w_2)}{N}$$

- In fact, this method extends to n-grams of any size:

$$P(w_1 \ldots w_n) = \frac{c(w_1 \ldots w_n)}{N}$$

A corpus of N tokens has N n-grams

- If we affix $n-1$ dummy symbols (<s1>, <s2> etc.) to the start or end of a text, then the text will contain N n-grams
- Result: a text of length N always has N n-grams (for any n)
- Example text: *a screaming comes across the sky* $(N = 6)$

Unigrams	Bigrams	Trigrams
a	<s1> *a*	<s1> <s2> *a*
screaming	*a screaming*	<s2> *a screaming*
comes	*screaming comes*	*a screaming comes*
across	*comes across*	*screaming comes across*
the	*across the*	*comes across the*
sky	*the sky*	*across the sky*

The `paste` command

- The paste command concatenates two vectors as strings

```
> paste("hello", "world")
[1] "hello world"
> paste("Today is", date())
[1] "Today is Sun Oct 24 21:10:24 2010"
```

- If the arguments are vectors, they are concatenated term-by-term, and recycled as needed

```
> paste(c("a","b"), c(1,2))
[1] "a 1" "b 2"

> paste("A", 1:5)
[1] "A 1" "A 2" "A 3" "A 4" "A 5"
```


Extracting bigrams in R using `paste`

```
# Create a list of six unigrams
> uni = c("a", "screaming", "comes", "across", "the", "sky")

# Remove the first element and add a period at the end
> uni2 <- c(uni[-1], ".")

> uni2
[1] "screaming" "comes" "across" "the" "sky" "."

# Find all bigrams using paste
> paste(uni, uni2)                 # Notice there are six
[1] "a screaming"     "screaming comes" "comes across"
[4] "across the"      "the sky"         "sky ."
```


Extracting trigrams in R using `paste`

```
> uni2
[1] "screaming" "comes" "across" "the" "sky" "."

# Again, remove the first element and add a period at the end
> uni3 <- c(uni2[-1], ".")

> uni3
[1] "comes"  "across" "the"    "sky"    "."      "."

# Find all trigrams using paste
> paste(uni, uni2, uni3)         # Notice there are six
[1] "a screaming comes"      "screaming comes across"
[3] "comes across the"       "across the sky"
[5] "the sky ."              "sky . ."
```


Bigram probabilities in the Bible

```
# Read in the text
> bible <- scan(what="c", sep="\n", file="bible-kjv.txt")
Read 74645 items

# Convert to lower case
> bible <- tolower(bible)

# Delete chapter:verse numbers like 1:1 when tokenizing
> tokens <- unlist(strsplit(bible, "[^a-z]+"))

# Remove empty tokens
> tokens <- tokens[tokens != ""]

# Compute N, the number of tokens
> length(tokens)
[1] 791842
```

Bigram probabilities in the Bible

```
# Remove the first element and add a period at the end
> tokens2 <- c(tokens[-1], ".")

# Note the length stays the same
> length(tokens2)
[1] 791842

# Create a sorted table of bigram type frequencies
> freq <- sort(table(paste(tokens, tokens2)), decreasing=T)

> head(freq)

  of the the lord  and the   in the   and he shall be
   11542     7035     6268     5031     2791     2461
```


Bigram probabilities in the Bible

```
> for (b in names(freq)[1:15]) {
   cat(b, freq[b], freq[b]/791842, "\n", sep="\t")
  };

of the      11542     0.01457614
the lord     7035     0.008884348
and the      6268     0.00791572
in the       5031     0.00635354
and he       2791     0.003524693
shall be     2461     0.003107943
to the       2152     0.002717714
all the      2144     0.002707611
and they     2086     0.002634364
unto the     2032     0.002566169
i will       1922     0.002427252
of israel    1697     0.002143104
for the      1675     0.002115321
the king     1659     0.002095115
said unto    1649     0.002082486
```


Trigram probabilities in the Bible

```
# Again, remove the first element and add a period at the end
> tokens3 <- c(tokens2[-1], ".")

# Create a vector of trigrams using paste
> trigrams <- paste(tokens, tokens2, tokens3)

# Create a sorted table of trigram type frequencies
> freq <- sort(table(trigrams), decreasing=T)

> head(freq)

trigrams
    of the lord      the son of   the children of
           1775            1450              1355
   the house of  saith the lord      the lord and
            883             854               816
```


Trigram probabilities in the Bible

```
> for (t in names(freq)[1:15]) {
   cat(t, freq[t], freq[t]/791842, "\n", sep="\t")
  };

of the lord         1775     0.002241609
the son of          1450     0.001831173
the children of     1355     0.0017112
the house of         883     0.001115121
saith the lord       854     0.001078498
the lord and         816     0.001030509
out of the           805     0.001016617
and i will           672     0.0008486542
children of israel   647     0.0008170822
the land of          617     0.0007791958
and the lord         571     0.0007211035
and all the          561     0.0007084747
the sons of          560     0.0007072118
and he said          510     0.0006440679
unto the lord        509     0.000642805
```

n-gram models of language

- An n-gram model is a statistical model of language in which the previous $n-1$ words are used to predict the next one

  Sue swallowed the large green . . .

- The term *n-gram* means 'sequence of length n'

n	Name	Example
1	unigram	*sky*
2	bigram	*the sky*
3	trigram	*across the sky*
4+	4-gram?	*comes across the sky*

- n-gram language models are often called just *language models*

Applications of word prediction

- Detecting typos that are real English words

  They are leaving in about fifteen *minuets* to go to her house.
  The study was conducted mainly *be* John Black.
  The design *an* construction of the system will take more than a year.
  Hopefully, all *with* continue smoothly in my absence.
  Can they *lave* him my messages?
  I need to *notified* the bank of [this problem.]
  He is trying to *fine* out.

- Avoiding unlikely speech recognition results
  - *The sign said key pout*
  - *Don't take a fence*
  - *Hi sealing*
  - *Ill eagle operations*
  - *Super fish'll dream*

Are words independent events?

- Why do we want to examine the previous $n-1$ words in order to predict the next one?

- In the case of coin flipping, the previous $n-1$ coin flips are useless in predicting the result of the next one, because each coin flip is an *independent event*

- In language, however, each word is highly dependent on the previous context

  weapons of mass . . .

- We therefore need the concept of *conditional probability* in order to make good predictions

Independence of events

- To say events A and B are *independent* means

$$P(A \cap B) = P(A)\ P(B)$$

- Coin toss example: the probability of two heads in a row is

$$\begin{aligned} P(h_1 \cap h_2) &= P(h_1)\ P(h_2) \\ &= 1/2 \times 1/2 = 1/4 \end{aligned}$$

- The probability of three heads in a row is

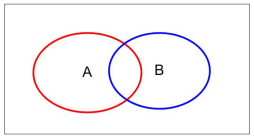
$$\begin{aligned} P(h_1 \cap h_2 \cap h_3) &= P(h_1)\ P(h_2)\ P(h_3) \\ &= 1/2 \times 1/2 \times 1/2 = 1/8 \end{aligned}$$

Conditional probability

- The *conditional probability* of an event A given that an event B has occurred (where $P(B) > 0$) is defined as

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

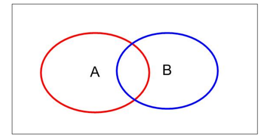
- This can be rewritten as the *product rule*: $P(A \cap B) = P(B)\ P(A|B)$
- If A and B are independent, then $P(A|B) = P(A)$



Example of the product rule

- Suppose it rains once every 10 days, and on rainy days you are 90% likely to carry an umbrella. What is the likelihood that you carry an umbrella in the rain on any given day?
- The product rule states $P(A \cap B) = P(B)\ P(A|B)$
- This gives us

$$\begin{aligned} P(umbrella \cap rain) &= P(rain)\ P(umbrella|rain) \\ &= 0.10 \times 0.90 \\ &= 0.09 \end{aligned}$$



More examples

- Two cards are drawn from a deck of 52 cards. What is the probability that both are queens?

$$\begin{aligned} P(Q_1 \cap Q_2) &= P(Q_1)\ P(Q_2|Q_1) \\ &= \tfrac{4}{52} \times \tfrac{3}{51} = \tfrac{1}{221} \end{aligned}$$

- A die is rolled; what is the probability of getting a 4, given that an even number was rolled?

$$\begin{aligned} P(F|E) &= \tfrac{P(F \cap E)}{P(E)} \\ &= \tfrac{\frac{1}{6}}{\frac{1}{2}} = \tfrac{1}{3} \end{aligned}$$

Predicting the next word

- According to the definition of conditional probability, the probability of seeing word w_n given the previous words $w_1, w_2, \ldots w_{n-1}$ is:

$$P(w_n|w_1 \ldots w_{n-1}) = \frac{P(w_1 \ldots w_n)}{P(w_1 \ldots w_{n-1})}$$

- Bigram example

$$P(lunch|eat) = \frac{P(eat\ lunch)}{P(eat)}$$

- Trigram example

$$P(lunch|to\ eat) = \frac{P(to\ eat\ lunch)}{P(to\ eat)}$$

Predicting the next word based on corpus counts

- We can *estimate* the conditional probability of seeing word w_n by using counts from a corpus:

$$P(w_n|w_1 \ldots w_{n-1}) = \frac{c(w_1 \ldots w_n)}{c(w_1 \ldots w_{n-1})}$$

- Bigram example

$$P(lunch|eat) = \frac{c(eat\ lunch)}{c(eat)}$$

- Trigram example

$$P(lunch|to\ eat) = \frac{c(to\ eat\ lunch)}{c(to\ eat)}$$

- This method is called *Maximum Likelihood Estimation* (MLE)

Predicting words based on Shakespeare

- What is the probability of seeing *the*, given that we've just seen *of*?

$$P(the|of) = \frac{c(of\ the)}{c(of)} = \frac{1496}{17481} = 0.086$$

- What is the probability of seeing *king*, given that we've just seen *the*?

$$P(king|the) = \frac{c(the\ king)}{c(the)} = \frac{877}{27378} = 0.032$$

Generating random sentences from WSJ

1. Sample sentence from a unigram language model:

   Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives.

2. Sample sentence from a bigram language model:

   Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the maj or 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.

3. Sample sentence from a trigram language model:

   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.

British food anyone?

- Statistical models of language implicitly encode both linguistic and cultural facts

- Bigram counts from the Google 1T web dataset

Chinese food	203,759
Mexican food	176,022
Indian food	120,449
Thai food	118,057
British food	36,444

- This is one reason Chomsky argued against statistical models

- Just because *Albanian food* doesn't appear in our corpus doesn't mean it doesn't exist

Smoothing

- MLE assigns zero probability to any n-gram not in the corpus
- This is too strict, because there are many perfectly good n-grams that just happen to not be in the corpus
- *Smoothing* is a way of assigning a small but non-zero probability to these "zero probability n-grams"
- Smoothing is also called *discounting* because the probabilities of the higher-probability n-grams are discounted a certain amount, and this amount is redistributed among the zero-probability n-grams
- This allows us to assign probabilities to utterances we have never seen before (e.g., *colorless green ideas sleep furiously*)

Add-one smoothing

- *Add-one* (*Laplace*) smoothing adds 1 to each count, then normalizes the denominator with the vocabulary size V
- For any n-gram a:

$$P'(a) = \frac{C(a) + 1}{N + V^n}$$

- In practice, add-one smoothing gives poor results because it assigns too much probability mass to unseen n-grams
- The problem: add-one smoothing assumes a uniform prior on events (i.e., that every n-gram is equally likely)
- In language, all n-grams are **not** equally likely (Zipf's Law)

Good-Turing discounting

- Good-Turing smoothes $P(a) = \frac{c_a}{N}$ to $P'(a) = \frac{c_a*}{N}$
- We discount the original MLE count c to $c* = (c+1)\frac{N_{c+1}}{N_c}$, where N_c is the number of N-grams that occur c times
- Example: Good-Turing discounted counts $c*$ for AP newswire bigrams (N_0 is V^2 minus all the bigrams we have seen):

c (MLE)	N_c	c* (GT)
0	74,671,100,000	0.0000270
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22
6	48,190	5.19
7	35,709	6.21
8	27,710	7.24
9	22,280	8.25

Backoff to bigrams and unigrams

- *Backoff* is another method for dealing with unseen n-grams
- One backoff method, due to Jelinek & Mercer (1980), computes the probability of a trigram as the sum of three weighted probabilities: trigram, bigram, and unigram

$$\begin{aligned} P'(w_i|w_{i-2}\ w_{i-1}) &= \lambda_1 P(w_i|w_{i-2}\ w_{i-1}) \\ &\quad +\lambda_2 P(w_i|w_{i-1}) \\ &\quad +\lambda_3 P(w_i) \end{aligned}$$

- Reasonable weights might be $\lambda_1 = 0.6$, $\lambda_2 = 0.3$, $\lambda_3 = 0.1$
- A better method is to compute separate weights for each bigram: $\lambda_1(w_{i-2}\ w_{i-1})$, $\lambda_2(w_{i-2}\ w_{i-1})$, $\lambda_3(w_{i-2}\ w_{i-1})$
- The larger $C(w_{i-2}\ w_{i-1})$ is, the more weight we give to λ_1

Evaluating language models: perplexity

- A good language model assigns high probability to test data
- We quantify this as *perplexity* (the lower the better), which is defined in terms of entropy: $PP(X) = 2^{H(X)}$
- For a test corpus $W = w_1 \ldots w_N$, the perplexity $PP(W)$ is

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

- For example, in the case of a bigram model we compute

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

Example of per-word perplexity

- Jurafsky & Martin trained unigram, bigram, and trigram grammars on 38 million words from the WSJ, using a 20,000 word vocabulary with backoff and GT discounting
- They then computed the perplexity of each of these models on a previously unseen test set of 1.5 million words

Model	Cross entropy $H(p,m)$	Perplexity $2^{H(p,m)}$
Unigram	9.91	962
Bigram	7.41	170
Trigram	6.77	109

- $H(p,m)$ is the *cross entropy* of model m of distribution p, and is an upper-bound estimate of the true entropy $H(p)$

Summary: language modeling

- Since words are not independent events, statistical language models employ *conditional probabilities*
- In particular, an n-*gram* language model uses the previous $n-1$ words to predict the next one

$$P(w_n|w_1 \ldots w_{n-1}) = \frac{P(w_1 \ldots w_n)}{P(w_1 \ldots w_{n-1})}$$

- We can *estimate* probabilities of particular word sequences using counts from a large corpus
- *Smoothing* (*discounting*) is used to assign small non-zero probabilities to n-grams that do not appear in the corpus
- Model m of distribution p can be evaluated in terms of its *cross entropy* $H(p,m)$ or *perplexity* $2^{H(p,m)}$ (lower is better)