Language Modeling

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
  - P(high winds tonite) > P(large winds tonite)
- Spelling Correction
  - The office is about fifteen minuets from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)

Why?

Probabilistic Language Modeling

Goal:

Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \dots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1,w_2,w_3,w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n | w_1,w_2 \dots w_{n-1})$ is called a **language model**.

How to compute P(W)

How to compute this joint probability:

- P(its, water, is, so, transparent, that)

Intuition: rely on the Chain Rule of Probability

Reminder: The Chain Rule

Recall the definition of conditional probabilities

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

Rewriting: **P(A,B) =** $P(A)\,P(B|A)$

$P(x_1, x_2 x_3 \ldots x_n) = P(x_1)\,P(x_2|x_1)\,P(x_3|x_1,x_2)\ldots$

$p(x_n|x_1 \ldots x_{n-1})$

Reminder: The Chain Rule

Recall the definition of conditional probabilities

p(B|A) = P(A,B)/P(A) Rewriting: **P(A,B) = P(A)P(B|A)**

The Chain Rule in General

The Chain Rule applied to compute joint probability of words in sentence

w@position 1

$P(w_1 w_2 \dots w_n) = \prod_i P(w_i \mid w_1 w_2 \dots w_{i-1})$

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i \mid \underline{w_1 w_2 \dots w_{i-1}})$$

P("its water is so transparent") = $\prod_i P(w_i \mid w_1 w_2 \dots w_{i-1})$

w_1 w_5

$= P(\text{its})\, P(\text{water} \mid \text{its}) \dots$

$P(\text{transparent} \mid \text{its water is so})$

$= P(w_1 = \text{its})\, P(w_2 = \text{water} \mid w_1 = \text{its})$

$P(w_3 = \text{is} \mid w_1 = \text{its}, w_2 = \text{water})$

$P(w_4 = \text{so} \mid w_1 = \text{its}, w_2 = \text{water}, w_3 = \text{is})$

$P(w_5 = \text{transparent} \mid w_1 = \text{its}, w_2 = \text{water}, w_3 = \text{is}, w_4 = \text{so})$

How to estimate these probabilities

Could we just count and divide?

<s>

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

Maximum Likelihood Estimates

How to estimate these probabilities

Could we just count and divide?

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

No! Too many possible sentences!

We'll never see enough data for estimating these

Markov Assumption

Simplifying assumption:

bigram: 2 words together

$$P(\text{the} \mid \cancel{\text{its water is so transparent}}\ \text{that}) \approx P(\text{the} \mid \text{that})$$

$$P(\text{the} \mid \cancel{\text{its water is so}}\ \text{transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

trigram model: 3 word units

Markov Assumption

Approximate each component in the product:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Bigram model

- Condition on the previous word:

$$P(w_i \mid w_1\, w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

```
texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november
```

N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language

- because language has **long-distance dependencies**:

  "The computer which I had just put into the machine room on the fifth floor crashed."

But we can often get away with N-gram models

Language Modeling

Estimating N-gram Probabilities

How do we get probabilities?

$$P(w_i \mid w_1\, w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

From observing frequencies in a training corpus

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

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

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$P(I \mid \langle s \rangle) = \frac{c(\langle s \rangle I)}{c(\langle s \rangle)} = 2/3$

$p(Sam \mid \langle s \rangle) = \frac{1}{3}$

$p(am \mid I) = 2/3$

$p(do \mid I) = 1/3$

More examples: 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

Raw bigram counts

Out of 9222 sentences

w_2

w_1

	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

Raw bigram probabilities

$$p(\text{want} \mid I) = \frac{c(I\ \text{want})}{c(I)}$$

Normalize by unigrams:

Result:

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>) = $p(I \mid \text{<s>})$

$\times\, p(\text{want} \mid I) = 0.33$

$p(\text{english} \mid \text{want}) =$

$p(\text{food} \mid \text{english}) =$

$p(\text{</s>} \mid \text{food})$

$= 0.000031$

What kinds of knowledge?

$P(\text{english} \mid \text{want}) = .0011$

$P(\text{chinese} \mid \text{want}) = .0065$

$P(\text{to} \mid \text{want}) = .66$

$P(\text{eat} \mid \text{to}) = .28$

$P(\text{food} \mid \text{to}) = 0$

$P(\text{want} \mid \text{spend}) = 0$

$P(\text{i} \mid \text{<s>}) = .25$

Practical Issues

When programming, we will handle probabilities in log space to avoid underflow. (This will be true for the rest of the class!)

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

Google N-Gram Release, August 2006

AUG
3

All Our N-gram are Belong to You

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,

...

That's why we decided 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.

Google N-Gram Release

```
serve as the incoming 92
serve as the incubator 99
serve as the independent 794
serve as the index 223
serve as the indication 72
serve as the indicator 120
serve as the indicators 45
serve as the indispensable 111
serve as the indispensible 40
serve as the individual 234
```

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

Language Modeling

Smoothing: Add-one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

P(w | denied the)

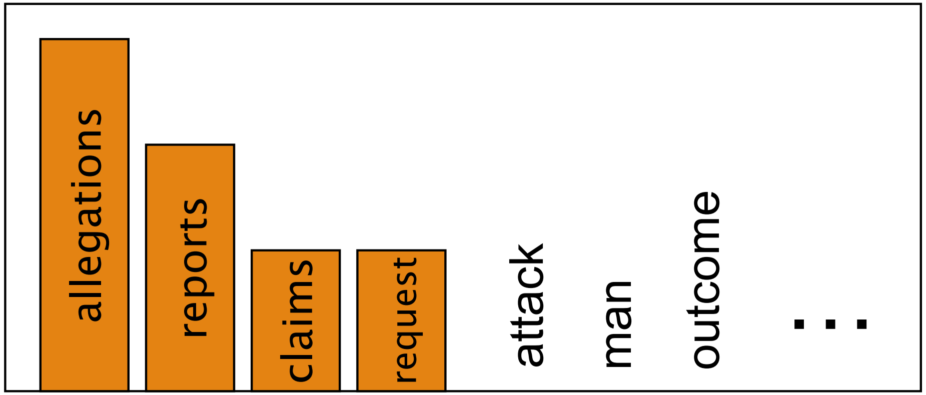
3 allegations

2 reports

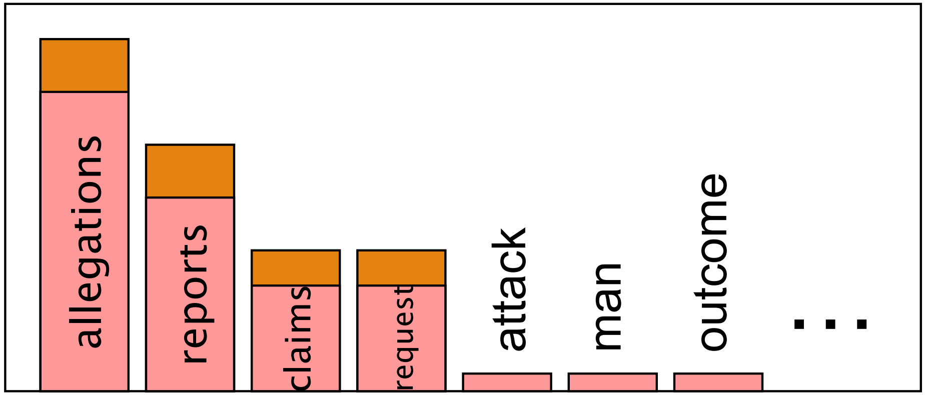
1 claims

1 request

7 total



Steal probability mass to generalize better



Add-one estimation

Also called Laplace smoothing

Pretend we saw each word one more time than we did

Just add one to all the counts!

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

V = size of vocabulary

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word "bagel" occurs 400 times in a corpus of a million words

What is the probability that a random word from some other text will be "bagel"?

MLE estimate:

This may be a bad estimate for some other corpus

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

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word "bagel" occurs 400 times in a corpus of a million words

What is the probability that a random word from some other text will be "bagel"?

MLE estimate is 400/1,000,000 = .0004

This may be a bad estimate for some other corpus

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

Berkeley Restaurant Corpus: 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 bigrams

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

	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

Compare with raw 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	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

Add-1 estimation is a blunt instrument

So add-1 isn't used for N-grams:

- We'll see better methods

But add-1 is used to smooth other NLP models

- For text classification
- In domains where the number of zeros isn't so huge.

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Backoff and Interpolation

Sometimes it helps to use **less** context

- Condition on less context for contexts you haven't learned much about

Backoff:

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

Interpolation:

- mix unigram, bigram, trigram

Interpolation works better

Linear Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Linear Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Let's set arbitrary weights: $\lambda_1 = 0.4$ $\lambda_2 = 0.6$

	i	want	to	eat	
i	5	827	0	9	1200
want	2	0	608	1	900
to	2	0	4	686	930
eat	0	0	2	0	39
	1200	900	930	39	

p(want|i) =

Linear Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Let's set arbitrary weights: $\lambda_1 = 0.4$ $\lambda_2 = 0.6$

	i	want	to	eat	
i	5	827	0	9	1200
want	2	0	608	1	900
to	2	0	4	686	930
eat	0	0	2	0	39
	1200	900	930	39	

p(want|to) =

Linear Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

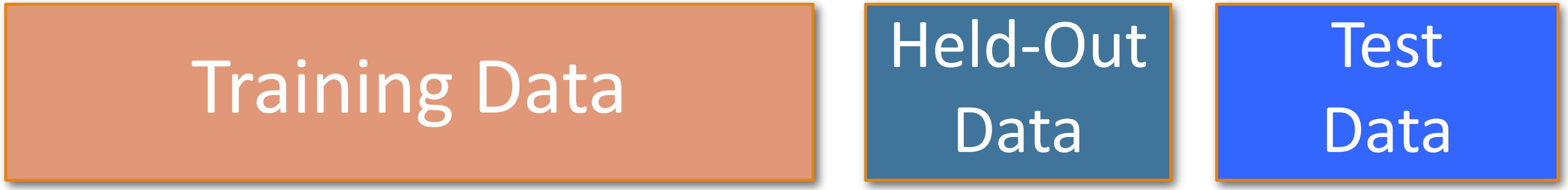
$$\sum_i \lambda_i = 1$$

Lambdas conditional on context:

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

How to set the lambdas?

Use a **held-out** corpus



Choose λs to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Unknown words: Open versus closed vocabulary tasks

If we know all the words in advanced

- Vocabulary V is fixed
- Closed vocabulary task

Often we don’t know this

- **Out Of Vocabulary** = OOV words
- Open vocabulary task

Instead: create an unknown word token <UNK>

- Training of <UNK> probabilities
  - Create a fixed lexicon L of size V
  - At text normalization phase, any training word not in L changed to <UNK>
  - Now we train its probabilities like a normal word
- At decoding time
  - If text input: Use UNK probabilities for any word not in training

Huge web-scale n-grams

How to deal with, e.g., Google N-gram corpus

Pruning
- Only store N-grams with count > threshold.
  - Remove singletons of higher-order n-grams
- Entropy-based pruning

Efficiency
- Efficient data structures like tries
- Bloom filters: approximate language models
- Store words as indexes, not strings
  - Use Huffman coding to fit large numbers of words into two bytes
- Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for Web-scale N-grams

"Stupid backoff" (Brants *et al*. 2007)

No discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \ \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4\,S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram Smoothing Summary

Add-1 smoothing:

- OK for text categorization, not for language modeling

The most commonly used method:

- Extended Interpolated Kneser-Ney (Intuition: instead of asking “How likely is w?”, ask “How likely is w to appear as a novel continuation?

For very large N-grams like the Web:

- Stupid backoff