

Natural Language Processing

- 3: Part-of-Speech Tagging. CRF
- 4: Language modeling. N-grams



CRF

Conditional Random Fields

Linear chain CRF

Which one is where (HMM / CRF)?

(1) X is input sequence, Y is sequence of tags

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_Y p(Y|X) \\ &= \operatorname{argmax}_Y p(X|Y)p(Y) \\ &= \operatorname{argmax}_Y \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1})\end{aligned}$$

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_{Y \in \mathcal{Y}} P(Y|X) \\ p(Y|X) &= \frac{\exp\left(\sum_{k=1}^K w_k F_k(X, Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^K w_k F_k(X, Y')\right)}\end{aligned}$$

Linear chain CRF

Which one is where (HMM / CRF)?

(1) X is input sequence, Y is sequence of tags

CRF

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_Y p(Y|X) \\ &= \operatorname{argmax}_Y p(X|Y)p(Y) \\ &= \operatorname{argmax}_Y \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1})\end{aligned}$$

HMM

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_{Y \in \mathcal{Y}} P(Y|X) \\ p(Y|X) &= \frac{\exp\left(\sum_{k=1}^K w_k F_k(X, Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^K w_k F_k(X, Y')\right)}\end{aligned}$$

What are F(X,Y)?

(1) Features:

$\mathbb{1}\{x_i = \textit{the}, y_i = \text{DET}\}$
 $\mathbb{1}\{y_i = \text{PROPN}, x_{i+1} = \textit{Street}, y_{i-1} = \text{NUM}\}$
 $\mathbb{1}\{y_i = \text{VERB}, y_{i-1} = \text{AUX}\}$

$$F_k(X, Y) = \sum_{i=1}^K f_k(y_{i-1}, p(Y|X)) = \frac{\exp\left(\sum_{k=1}^K w_k F_k(X, Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^K w_k F_k(X, Y')\right)}$$

Feature Templates

$\langle y_i, x_i \rangle, \langle y_i, y_{i-1} \rangle, \langle y_i, x_{i-1}, x_{i+2} \rangle$

f_{3743} : $y_i = \text{VB}$ and $x_i = \text{back}$

f_{156} : $y_i = \text{VB}$ and $y_{i-1} = \text{MD}$

f_{99732} : $y_i = \text{VB}$ and $x_{i-1} = \text{will}$ and $x_{i+2} = \text{bill}$

x_i contains a particular prefix (perhaps from all prefixes of length ≤ 2)

x_i contains a particular suffix (perhaps from all suffixes of length ≤ 2)

x_i 's word shape

x_i 's short word shape

Inference and Training for CRFs

inference == decoding

(1) Can we use Viterbi for decoding?

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_{Y \in \mathcal{Y}} P(Y|X) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \frac{1}{Z(X)} \exp \left(\sum_{k=1}^K w_k F_k(X, Y) \right) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \exp \left(\sum_{k=1}^K w_k \sum_{i=1}^n f_k(y_{i-1}, y_i, X, i) \right) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \sum_{k=1}^K w_k \sum_{i=1}^n f_k(y_{i-1}, y_i, X, i) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \sum_{i=1}^n \sum_{k=1}^K w_k f_k(y_{i-1}, y_i, X, i)\end{aligned}$$

Homework

optional: apply Viterbi algorithm to find POS tags

(1) a bear likes sweet honey

(2) [DT] [ADJ, VERB, NOUN] [VERB, NOUN] [ADJ, NOUN] [ADJ, NOUN]

	ADJ	NOUN	VERB	DT
ADJ	0.4	0.4	0.2	0.1
NOUN	0.2	0.4	0.4	0.1
VERB	0.1	0.6	0.3	0.1
DT	0.29	0.6	0.1	0.01
[START]	0.1	0.4	0.1	0.4

	a	bear	likes	fly	honey	sweet
ADJ	0.2	0.1	0.1	0.1	0.1	0.4
NOUN	0.1	0.2	0.2	0.2	0.2	0.1
VERB	0.1	0.2	0.3	0.2	0.1	0.1
DT	0.5	0.1	0.05	0.1	0.1	0.1

Language Modeling.

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
- Spell Correction
 - The office is about fifteen minuets from my house
 - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- Speech Recognition
 - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
- + Summarization, question-answering, etc., etc.!!

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**.

Better: **the grammar** But **language model** or **LM** is standard

How to compute $P(W)$

How to compute this joint probability:

- $P(\text{its, water, is, so, transparent, that})$

Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

Recall the definition of conditional probabilities

$$p(B | A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B | A)$$

More variables:

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

The Chain Rule in General

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

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

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

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

How to estimate these probabilities

Could we just count and divide?

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

No! Too many possible sentences!

We'll never see enough data for estimating these

Markov Assumption



Andrei Markov

Simplifying assumption:

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

Or maybe

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

Markov Assumption (assignment for you)

In other words, we approximate each component in the product

Markov Assumption (assignment for you)

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

In other words, we approximate each component in the product

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

Simplest case: Unigram model

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

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:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

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

Estimating N-gram Probabilities

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

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

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

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

	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

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

$P(<s> \text{ I want english food } </s>) =$

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

$\times P(\text{want} | \text{I})$

$\times P(\text{english} | \text{want})$

$\times P(\text{food} | \text{english})$

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

$= .000031$

What kinds of knowledge?

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

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

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

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

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

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

$$P(i | \langle s \rangle) = .25$$

Practical Issues

We do everything in log space

- Avoid underflow
- (also adding is faster than multiplying)

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

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 indispensable 40
serve as the individual 234

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

Google Book N-grams

<http://ngrams.googlelabs.com/>

Evaluation and Perplexity

How to evaluate N-gram models

"Extrinsic (in-vivo) Evaluation"

To compare models A and B

1. Put each model in a real task
 - Machine Translation, speech recognition, etc.
2. Run the task, get a score for A and for B
 - How many words translated correctly
 - How many words transcribed correctly
3. Compare accuracy for A and B

Intrinsic (in-vitro) evaluation

Extrinsic evaluation not always possible

- Expensive, time-consuming
- Doesn't always generalize to other applications

Intrinsic evaluation: **perplexity**

- Directly measures language model performance at predicting words.
- Doesn't necessarily correspond with real application performance
- But gives us a single general metric for language models
- Useful for large language models (LLMs) as well as n-grams

Training sets and test sets

We train parameters of our model on a **training set**.

We test the model's performance on data we haven't seen.

- A **test set** is an unseen dataset; different from training set.
- Intuition: we want to measure generalization to unseen data
- An **evaluation metric** (like **perplexity**) tells us how well our model does on the test set.

Choosing training and test sets

- If we're building an LM for a specific task
 - The test set should reflect the task language we want to use the model for
- If we're building a general-purpose model
 - We'll need lots of different kinds of training data
 - We don't want the training set or the test set to be just from one domain or author or language.

Training on the test set

We can't allow test sentences into the training set

- Or else the LM will assign that sentence an artificially high probability when we see it in the test set
- And hence assign the whole test set a falsely high probability.
- Making the LM look better than it really is

This is called “Training on the test set”

Bad science!

Dev sets

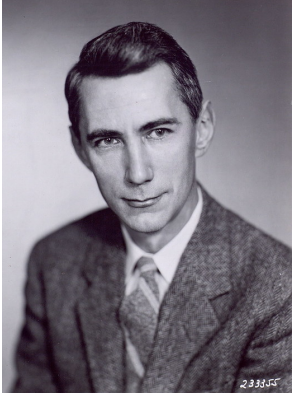
- If we test on the test set many times we might implicitly tune to its characteristics
 - Noticing which changes make the model better.
- So we run on the test set only once, or a few times
- That means we need a third dataset:
 - A **development test set** or, **devset**.
 - We test our LM on the devset until the very end
 - And then test our LM on the **test set** once

Intuition of perplexity as evaluation metric: How good is our language model?

Intuition: A good LM prefers "real" sentences

- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?

Intuition of perplexity 2: Predicting upcoming words



Claude Shannon

The Shannon Game: **How well can we predict the next word?**

- Once upon a _____
- That is a picture of a _____
- For breakfast I ate my usual _____

time	0.9
dream	0.03
midnight	0.02
...	
and	1e-100

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs

Intuition of perplexity 3: The best language model is one that best predicts the entire unseen test set

- We said: a good LM is one that assigns a higher probability to the next word that actually occurs.
- Let's generalize to all the words!
 - The best LM assigns high probability to the entire test set.
- When comparing two LMs, A and B
 - We compute $P_A(\text{test set})$ and $P_B(\text{test set})$
 - The better LM will give a higher probability to (=be less surprised by) the test set than the other LM.

Intuition of perplexity 4: Use perplexity instead of raw probability

- Probability depends on size of test set
 - Probability gets smaller the longer the text
 - Better: a metric that is **per-word**, normalized by length
- **Perplexity** is the inverse probability of the test set, normalized by the number of words

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

Intuition of perplexity 5: the inverse

Perplexity is the **inverse** probability of the test set, normalized by the number of words

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)

Probability range is $[0,1]$, perplexity range is $[1,\infty]$

Minimizing perplexity is the same as maximizing probability

Intuition of perplexity 6: N-grams

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

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

Chain rule:

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

Bigrams:

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

Intuition of perplexity 7: Weighted average branching factor

Perplexity is also the **weighted average branching factor** of a language.

Branching factor: number of possible next words that can follow any word

Example: Deterministic language $L = \{\text{red}, \text{blue}, \text{green}\}$

Branching factor = 3 (any word can be followed by red, blue, green)

Now assume LM A where each word follows any other word with equal probability $\frac{1}{3}$

Given a test set $T = \text{"red red red red blue"}$

$$\text{Perplexity}_A(T) = P_A(\text{red red red red blue})^{-1/5} = ((\frac{1}{3})^5)^{-1/5} = (\frac{1}{3})^{-1} = 3$$

But now suppose red was very likely in training set, such that for LM B:

- $P(\text{red}) = .8 \quad p(\text{green}) = .1 \quad p(\text{blue}) = .1$

We would expect the probability to be higher, and hence the perplexity to be smaller:

$$\text{Perplexity}_B(T) = P_B(\text{red red red red blue})^{-1/5}$$

$$= (.8 * .8 * .8 * .8 * .1)^{-1/5} = .04096^{-1/5} = .527^{-1} = 1.89$$

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Generalization and zeros

The Shannon Visualization Method

Choose a random bigram

($\langle s \rangle$, w) according to its probability

Now choose a random bigram

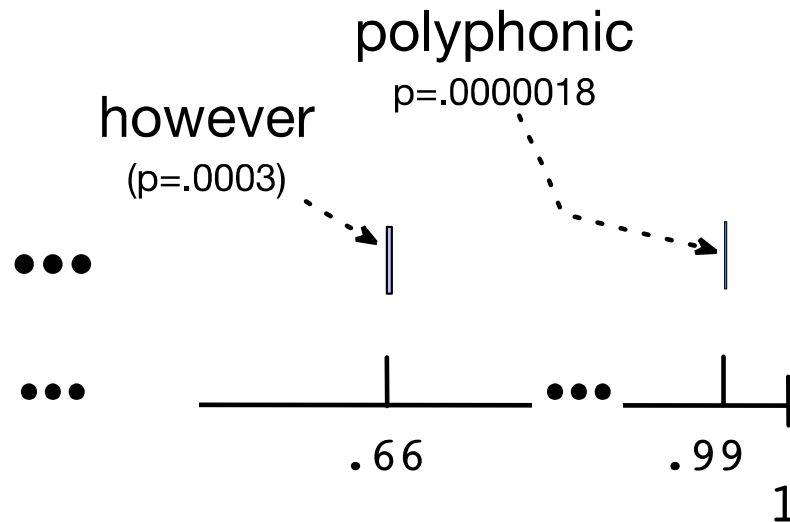
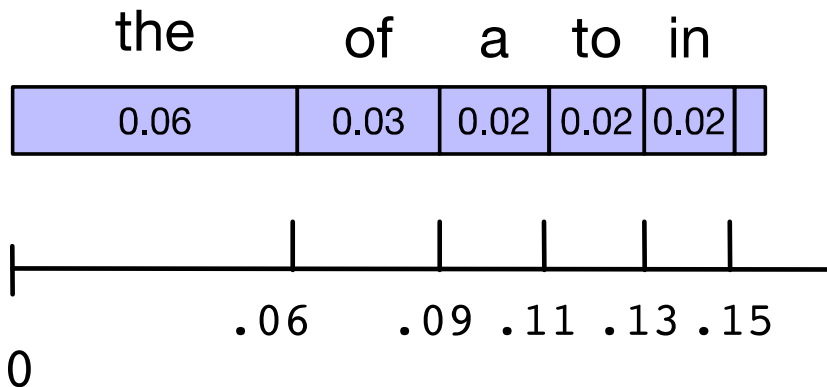
(w , x) according to its probability

And so on until we choose $\langle /s \rangle$

Then string the words together

$\langle s \rangle$ I
I want
want to
to eat
eat Chinese
Chinese food
food $\langle /s \rangle$
I want to eat Chinese food

Sampling a word from a distribution



Note: there are other sampling methods

Used for neural language models

Many of them avoid generating words from the very unlikely tail of the distribution

We'll discuss when we get to neural LM decoding:

- Temperature sampling
- Top-k sampling
- Top-p sampling

Approximating Shakespeare

1

gram

–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have

–Hill he late speaks; or! a more to leg less first you enter

2

gram

–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

–What means, sir. I confess she? then all sorts, he is trim, captain.

3

gram

–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

–This shall forbid it should be branded, if renown made it empty.

4

gram

–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;

–It cannot be but so.

Shakespeare as corpus

$N=884,647$ 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 worse: What's coming out looks like Shakespeare because it *is* Shakespeare

The Wall Street Journal is not Shakespeare

1
gram

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

2
gram

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

3
gram

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

Choosing training data

If task-specific, use a training corpus that has a similar genre to your task.

- If legal or medical, need lots of special-purpose documents

Make sure to cover different kinds of dialects and speaker/authors.

- Example: *African-American Vernacular English (AAVE)*
 - One of many varieties that can be used by African Americans and others
 - Can include the auxiliary verb **finna** that marks immediate future tense:
 - "My phone finna die"

The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
- One kind of generalization: Zeros!
 - Things that don't ever occur in the training set
 - But occur in the test set

Zeros

Training set:

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

- Test set
 - ... denied the offer
 - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

Zero probability bigrams

Bigrams with zero probability

- mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

Smoothing: Add-one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

$P(w \mid \text{denied the})$

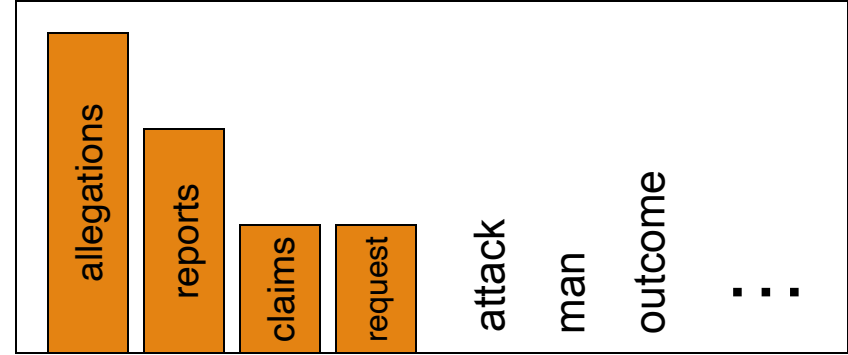
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

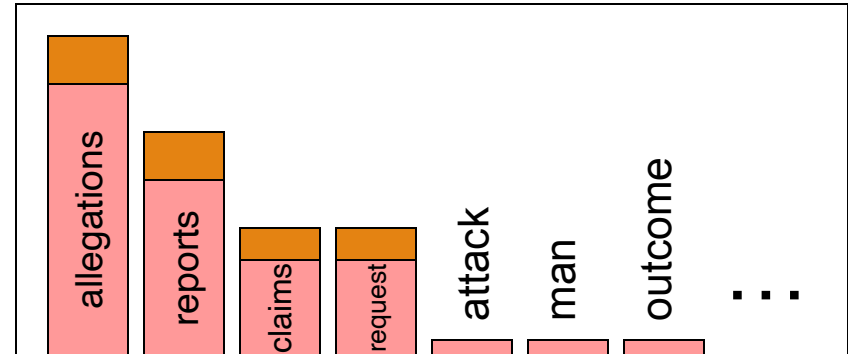
1.5 reports

0.5 claims

0.5 request

2 other

7 total



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

Add-1 estimate:
$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

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.

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

References

<https://web.stanford.edu/~jurafsky/slp3>

(1) Chapter 3 from the SLP book

(1) <https://web.stanford.edu/~jurafsky/slp3/3.pdf>

(2) Slides from here:

(1) https://web.stanford.edu/~jurafsky/slp3/slides/3_LM_2024.pptx