CS498JH: Introduction to NLP (Fall 2012)
http://cs.illinois.edu/class/cs498jh

Lecture 3: Smoothing

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Office Hours: Wednesday, 12:15-1:15pm

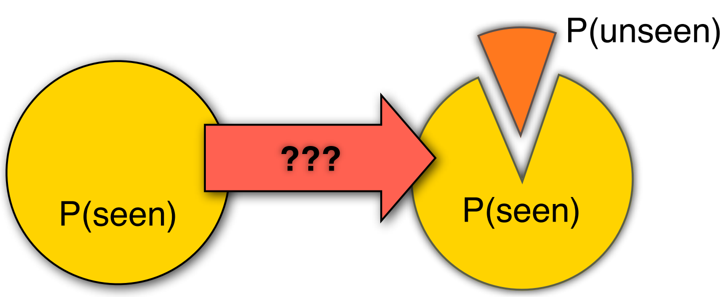
Wednesday's key concepts

N-gram language models
Independence assumptions
Relative frequency estimation
Unseen events
Zipf's law

Today's lecture

How can we design language models*
that can deal with previously unseen events?

*actually, probabilistic models in general



Parameter estimation (training)

Parameters: the actual probabilities
$P(w_i = \text{'the'} \mid w_{i-1} = \text{'on'}) = ???$

We need (a large amount of) text as training data
to estimate the parameters of a language model.

The most basic estimation technique:
relative frequency estimation (= counts)
$P(w_i = \text{'the'} \mid w_{i-1} = \text{'on'}) = f(\text{'on the'}) \,/\, f(\text{'on'})$

This assigns *all* probability mass to events in the training corpus

How do we evaluate models?

Define an evaluation metric (scoring function).

We will want to measure how similar the predictions of the model are to real text.

Train the model on a *'seen'* training set

Perhaps: tune some parameters based on held-out data (disjoint from the training data, meant to emulate unseen data)

Test the model on an *unseen* test set

(usually from the same source (e.g. WSJ) as the training data)
Test data must be disjoint from training and held-out data
Compare models by their scores.

Testing: unseen events will occur

Recall the Shakespeare example:

Only 30,000 word types occurred.
Any word that does not occur in the training data has zero probability!

Only 0.04% of all possible bigrams occurred.
Any bigram that does not occur in the training data has zero probability!

Dealing with unseen events

Relative frequency estimation assigns all probability mass to events in the training corpus

But we need to reserve *some* probability mass to events that don't occur in the training data

Unseen events = new words, new bigrams

Important questions:

What possible events are there?
How much probability mass should they get?

What unseen events may occur?

Simple distributions:

$$P(X = x)$$

(e.g. unigram models)

The outcome x may be unseen (i.e. completely unknown):

We need to reserve mass in $P(X)$.

What outcomes x are possible?
How much mass should they get?

What unseen events may occur?

Simple conditional distributions:

$$P(X = x \mid Y = y)$$

(e.g. bigram models)

The outcome x may be known,
but not in the context of y:
We need to reserve mass in $P(X \mid Y=y)$

The conditioning variable y may not be known:
We have no $P(X \mid Y=y)$ distribution.
We need to drop y and use $P(X)$ instead.

What unseen events may occur?

Complex conditional distributions

$$P(X = x \mid Y = y, Z = z)$$

(e.g. trigram models)

The outcome x may be known,
but not in the context of y,z:
We need to reserve mass in $P(X \mid Y=y,Z=z)$

The joint conditioning event $(Y=y, Z=z)$ may be unknown:
We have no $P(X \mid Y=y, Z=z)$ distribution.
We need to drop z and use $P(X \mid Y=y)$ instead.

Examples

Training data: The wolf was an endangered species
Test data: The wallaby is endangered

Unigram	**Bigram**	**Trigram**
P(the)	P(the | <s>)	P(the | <s>)
× P(wallaby)	× P(wallaby | the)	× P(wallaby | the, <s>)
× P(is)	× P(is | wallaby)	× P(is | wallaby, the)
× P(endangered)	× P(endangered | is)	× P(endangered | is, wallaby)

- **Case 1:** P(wallaby), P(wallaby | the), P(wallaby | the, <s>):
  What is the probability of an unknown word (in any context)?
- **Case 2:** P(endangered | is)
  What is the probability of a known word in a known context,
  if that word hasn't been seen in that context?
- **Case 3:** P(is | wallaby) P(is | wallaby, the) P(endangered | is, wallaby):
  What is the probability of a known word in an unseen context?

Smoothing: Reserving mass in P(X) for unseen events

Dealing with unknown words: The simple solution

Training:

- Assume a fixed vocabulary
  (e.g. all words that occur at least 5 times in the corpus)
- Replace all other words by a token <UNK>
- Estimate the model on this corpus.

Testing:

- Replace all unknown words by <UNK>
- Run the model.

Dealing with unknown events

Use a different estimation technique:

- Add-1(Laplace) Smoothing
- Good-Turing Discounting

Idea: *Replace MLE estimate* $P(w) = \frac{C(w)}{N}$

Combine a complex model with a simpler model:

- Linear Interpolation
- Modified Knesser-Ney smoothing

Idea: use bigram probabilities of w_i $P(w_i|w_{i-1})$
to calculate trigram probabilities of w_i $P(w_i|w_{i-n}...w_{i-1})$

Add-1 (Laplace) smoothing

Assume every (seen or unseen) event occurred once more than it did in the training data.

Example: unigram probabilities
Estimated from a corpus with N tokens and a vocabulary (number of word types) of size V.

$$\text{MLE} \quad P(w_i) = \frac{C(w_i)}{\sum_j C(w_j)} = \frac{C(w_i)}{N}$$

$$\text{Add One} \quad P(w_i) = \frac{C(w_i)+\mathbf{1}}{\sum_j (C(w_j)+\mathbf{1})} = \frac{C(w_i)+\mathbf{1}}{N+\mathbf{V}}$$

Bigram counts

Original:

	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

Smoothed:

	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

Bigram probabilities

Original:

	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

Smoothed:

	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

Problem:
Add-one moves too much probability mass from seen to unseen events!

Reconstituting the counts

We can "reconstitute" pseudo-counts for our training set of size N from our estimate:

Unigrams:

$$\begin{aligned} c_i^* &= P(w_i) \cdot N \\ &= \frac{C(w_i)+1}{N+V} \cdot N \\ &= (C(w_i)+1) \cdot \frac{N}{N+V} \end{aligned}$$

Bigrams:

$$\begin{aligned} c^*(w_{i-1}w_i) &= P(w_{i-1}w_1) \cdot C(w_{i-1}) \\ &= \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V} \cdot C(w_{i-1}) \end{aligned}$$

Reconstituted Bigram counts

Original:

	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

Reconstituted:

	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

Summary: Add-One smoothing

Advantage:
Very simple to implement

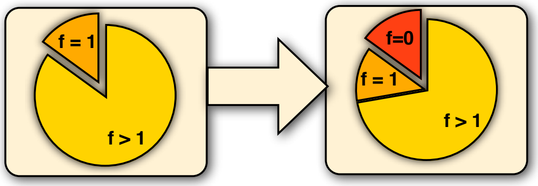
Disadvantage:
Takes away too much probability mass from seen events.
Assigns too much total probability mass to unseen events.

The Shakespeare example
(V = 30,000 word types; '*the*' occurs 25,545 times)
Bigram probabilities for 'the …':

$$P(w_i|w_{i-1} = the) = \frac{C(the\ w_i)+\mathbf{1}}{25,545+\mathbf{30,000}}$$

Good-Turing smoothing

Idea: Use **total frequency of events that occur only once** to estimate **how much mass to shift to unseen events**



	$P(seen)$	+	$P(unseen)$	=	1
MLE	$\frac{N}{N}$	+	0	=	1
Good Turing	$\frac{2\cdot N_2+\ldots+m\cdot N_m}{\sum_{i=1}^m i\cdot N_i}$	+	$\frac{1\cdot N_1}{\sum_{i=1}^m i\cdot N_i}$	=	$\frac{\sum_{i=1}^m i\cdot N_i}{\sum_{i=1}^m i\cdot N_i}$

- N_c: number of *event types* that occur *c* times *(can be counted)*
- N_1: number of *event types that occur once*
- $N = 1N_1+\ldots+mN_m$: total number of observed event tokens

Good-Turing smoothing

Reassign the **probability mass of all events that occur *n* times** in the training data to **all events that occur *n-1* times**.

- N_n events occur n times, with a total frequency of $n\cdot N_n$

The probability mass of all words that appear n-1 times becomes:

$$\sum_{w:C(w)=n-1} P_{GT}(w) = \sum_{w':C(w')=n} P_{MLE}(w') = \sum_{w':C(w')=n} \frac{n}{N} = \frac{n\cdot N_n}{N}$$

There are N_{n-1} words w that occur n-1 times in the training data.
Good-Turing replaces the original count c_{n-1} of w with a new count c^*_{n-1}:

$$c^*_{n-1} = \frac{n\cdot N_n}{N_{n-1}}$$

Good-Turing smoothing

The **Maximum Likelihood estimate** of the probability of a word *w* that occurs n-1 times:

$$P_{MLE}(w) = \frac{c_{n-1}}{N} = \frac{n-1}{N}$$

The **Good-Turing estimate** of the probability of a word *w* that occurs n-1 times:

$$P_{GT}(w) = \frac{c^*_{n-1}}{N} = \frac{\left(\frac{n\cdot N_n}{N_{n-1}}\right)}{N} = \frac{n\cdot N_n}{N\cdot N_{n-1}}$$

Problems with Good-Turing

Problem 1:
What happens to the most frequent event?

Problem 2:
We don't observe events for every n.

Variant: Simple Good-Turing
Replace N_n with a fitted function $f(n)$:

$$f(n) = a + b\log(n)$$

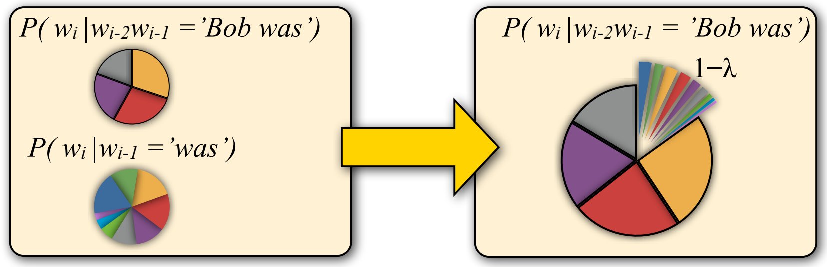
- Set a,b so that $f(n) \cong N_n$ for known values.
  Use c_n^* only for small n

Smoothing: Reserving mass in P(X |Y) for unseen events

Linear Interpolation (1)

We don't see "*Bob was reading*", but we see "__ *was reading*".
We estimate *P(reading | 'Bob was') =0* but *P(reading | 'was') > 0*

Use *(n-1)*-gram probabilities to smooth *n*-gram probabilities:



$$\underbrace{\tilde{P}_{LI}(w_i|w_{i-n}w_{i-n+1}\ldots w_{i-2}w_{i-1})}_{\text{smoothed n-gram}} = \lambda\,\underbrace{\hat{P}(w_i|w_{i-n}w_{i-n+1}\ldots w_{i-2}w_{i-1})}_{\text{unsmoothed n-gram}} + (1-\lambda)\,\underbrace{\tilde{P}_{LI}(w_i|w_{i-n+1}\ldots w_{i-2}w_{i-1})}_{\text{smoothed (n-1)-gram}}$$

Linear Interpolation (2)

We've never seen "*Bob was reading*",
but we might have seen "__ *was reading*",
and we've certainly seen "__ *reading*" (or <UNK>)

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

$$\text{for } \lambda_1 + \lambda_2 + \lambda_3 = 1$$

Interpolation: Setting the λs

Method A: Held-out estimation

Divide data into training and held-out data.
Estimate models on training data.
Use held-out data (and some optimization technique) to find the λ that gives best model performance.
(We'll talk about evaluation later)

λ can also depend on $w_{i-n}\ldots w_{i-1}$

Method B:

λ is some function of the frequencies of $w_{i-n}\ldots w_{i-1}$

Absolute discounting

Subtract a constant factor $D < 1$ from each nonzero *n*-gram count, and interpolate with $P_{AD}(w_i \mid w_{i-1})$:

non-zero if trigram $w_{i-2}w_{i-1}w_i$ is seen

$$P_{AD}(w_i|w_{i-1},w_{i-2}) = \frac{\max(C(w_{i-2}w_{i-1}w_i) - D, 0)}{C(w_{i-2}w_{i-1})} + (1-\lambda)P_{AD}(w_i|w_{i-1})$$

If S seen word types occur after $w_{i-2}\, w_{i-1}$ in the training data, this reserves the probability mass $P(U) = (S \times D)/C(w_{i-2}w_{i-1})$ to be computed according to $P(w_i \mid w_{i-1})$. Set:

$$(1-\lambda) = P(U) = \frac{S \cdot D}{C(w_{i-2}w_{i-1})}$$

N.B.: with N_1, N_2 the number of *n*-grams that occur once or twice, $D = N_1/(N_1+2N_2)$ works well in practice

Kneser-Ney smoothing

Observation: *"San Francisco"* is frequent, but *"Francisco"* only occurs after "*San*".

Solution: the unigram probability $P(w)$ should not depend on the frequency of w, but on the number of contexts in which w appears

$N_{+1}(\bullet w)$: number of contexts in which w appears
= number of word types w' which precede w
$N_{+1}(\bullet\bullet) = \sum_{w'} N_{+1}(\bullet w')$

Kneser-Ney smoothing: Use absolute discounting, but use $P(w) = N_{+1}(\bullet w)/N_{+1}(\bullet\bullet)$

Modified Kneser-Ney smoothing: Use different *D for bigrams and trigrams (Chen & Goodman '98)*

To recap….

Today's key concepts

Dealing with unknown words
Dealing with unseen events
Good-Turing smoothing
Linear Interpolation
Absolute Discounting
Kneser-Ney smoothing

Today's reading:
Jurafsky and Martin, Chapter 4, sections 1-4