N-grams
&
Smoothing

Chapter 4

All materials, adapted and used with permission from Bridget McInnes and Ted Pedersen.

Overview

- Recap of N-gram Modeling
  - Markov Assumption
  - How to Create Relative Frequency Tables?
  - How to Compute Probability of an Utterance
- How to Generate Sentences
  - Methods, Examples, Summarized Steps
- Zeros & Sparsity
  - Zeros
  - Sparsity
  - Recap MLE
- Smoothing
  - Zipfian Distribution
  - Laplace Smoothing (Add-1 smoothing)
  - Good-Turing
  - Backoff
    - Smoothing for Web-scale N-grams
  - Smoothing Summary

Added by Dr. Liao

Markov assumption

- Estimate the conditional probability of the next word without looking too far in the past

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

Recap of N-gram Modeling

Bigram: $$P(x_1^n) = \prod_{k=1}^{n} P\left(x_k \middle| x_1^{k-1}\right) = \prod_{k=1}^{n} P(x_k | x_{k-1})$$

Trigram: $$P(x_1^n) = \prod_{k=1}^{n} P\left(x_k \middle| x_1^{k-1}\right) = \prod_{k=1}^{n} P\left(x_k \middle| x_{k-2}^{k-1}\right)$$

4-gram: $$P(x_1^n) = \prod_{k=1}^{n} P\left(x_k \middle| x_1^{k-1}\right) = \prod_{k=1}^{n} P\left(x_k \middle| x_{k-3}^{k-1}\right)$$

Recap of Relative Frequency

For N-gram, $P(x_1 \dots x_n) = \prod_{k=1}^{n} P(xk|x_1 \dots xk_{-1})$

$\approx \prod_{k=1}^{n} P(xk|xk_{-N+1,\dots,}x_{k-1})$

$$P(x_1^n) = \prod_{k=1}^{n} P\left(x_k \middle| x_1^{k-1}\right) = \prod_{k=1}^{n} P\left(x_k \middle| x_{k-N+1}^{k-1}\right)$$

$$P\left(x_k \middle| x_{k-N+1}^{k-1}\right) = \frac{freq(x_{k-N+1}^{k-1}\, x_k)}{freq(x_{k-N+1}^{k-1})}$$

this ratio is called relative frequency

The equations were added by Dr. Liao

Recap of N-gram Modeling Example

Let's see the first simple example from (Dr. Seuss' story):

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

Now we are going to use the Bigram model

$$P(x_k | x_{k-1}) = \frac{freq(x_{k-1} x_k)}{freq(x_{k-1})}$$

$P(\text{ am } | \text{ I }) = P(\text{ I am }) / P(\text{I}) = 2/3$

$P(\text{ I } | \text{<s>}) = 2/3$ $P(\text{ Sam } | \text{<s>}) = 1/3$

$P(\text{ </s> } | \text{Sam }) = 1/2$ $P(\text{ Sam } | \text{am }) = 1/2$ $P(\text{ do} | \text{I }) = 1/3$

Added by Dr. Liao

How to Create Relative Frequency Tables?

Example:

CORPUS

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

Added by Dr. Liao

How to Create Relative Frequency Tables?

CORPUS

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

Unigram Table

Bigram Table

Added by Dr. Liao

How to Create Relative Frequency Tables?

CORPUS

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

Unigram Table

I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
3	2	2	1	1	1	1	1	1	1	3	3

Bigram Table

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I		2		1								
am			1									1
Sam	1											1
do					1							
not						1						
like							1					
green								1				
eggs									1			
and										1		
ham												1
<s>	2		1									
</s>												

Added by Dr. Liao

How to Create Relative Frequency Tables?

Unigram Table

I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
3	2	2	1	1	1	1	1	1	1	3	3

Bigram Table

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I	0	2	0	1	0	0	0	0	0	0	0	0
am	0	0	1	0	0	0	0	0	0	0	0	1
Sam	1	0	0	0	0	0	0	0	0	0	0	1
do	0	0	0	0	1	0	0	0	0	0	0	0
not	0	0	0	0	0	1	0	0	0	0	0	0
like	0	0	0	0	0	0	1	0	0	0	0	0
green	0	0	0	0	0	0	0	1	0	0	0	0
eggs	0	0	0	0	0	0	0	0	1	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0
ham	0	0	0	0	0	0	0	0	0	0	0	1
<s>	2	0	1	0	0	0	0	0	0	0	0	0
</s>	0	0	0	0	0	0	0	0	0	0	0	0

CORPUS

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

Freq(am|I)
= Freq(I am)/Freq(I) =

Added by Dr. Liao

Relative Frequencies

How to Create Relative Frequency Tables?

Unigram Table

I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
3	2	2	1	1	1	1	1	1	1	3	3

Bigram Table

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I	0	2	0	1	0	0	0	0	0	0	0	0
am	0	0	1	0	0	0	0	0	0	0	0	1
Sam	1	0	0	0	0	0	0	0	0	0	0	1
do	0	0	0	0	1	0	0	0	0	0	0	0
not	0	0	0	0	0	1	0	0	0	0	0	0
like	0	0	0	0	0	0	1	0	0	0	0	0
green	0	0	0	0	0	0	0	1	0	0	0	0
eggs	0	0	0	0	0	0	0	0	1	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0
ham	0	0	0	0	0	0	0	0	0	0	0	1
<s>	2	0	1	0	0	0	0	0	0	0	0	0
</s>	0	0	0	0	0	0	0	0	0	0	0	0

CORPUS

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

Freq(am|I)
= Freq(I am)/Freq(I) =2/3

Added by Dr. Liao

How to Create Relative Frequency Tables?

Unigram Table

I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
3	2	2	1	1	1	1	1	1	1	3	3

Relative Frequencies

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I	0	2/3	0	1/3	0	0	0	0	0	0	0	0
am	0	0	1/2	0	0	0	0	0	0	0	0	1/2
Sam	1/2	0	0	0	0	0	0	0	0	0	0	1/2
do	0	0	0	0	1	0	0	0	0	0	0	0
not	0	0	0	0	0	1	0	0	0	0	0	0
like	0	0	0	0	0	0	1	0	0	0	0	0
green	0	0	0	0	0	0	0	1	0	0	0	0
eggs	0	0	0	0	0	0	0	0	1	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0
ham	0	0	0	0	0	0	0	0	0	0	0	1
<s>	2/3	0	1/3	0	0	0	0	0	0	0	0	0
</s>	0	0	0	0	0	0	0	0	0	0	0	0

CORPUS

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

Freq(am|I)
= Freq(I am)/Freq(I) =2/3

Added by Dr. Liao

How to Compute Probability of an Utterance?

- Given an n-gram language model

P(<s> I am Sam </s>) =

P(<s>) *
P(I|<s>) *
P(am| I) *
P(Sam | am) *
P(</s>| Sam)

This is called the *chain rule of probability* using *the markov assumption*

Added by Dr. Liao

P(<s> I am Sam </s>) =

- P(<s>) *
- P(I|<s>) *
- P(am| I) *
- P(Sam | am) *
- P(</s>| Sam)

Bigram Relative Frequency Table (from training set)

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I	0	2/3	0	1/3	0	0	0	0	0	0	0	0
am	0	0	1/2	0	0	0	0	0	0	0	0	1/2
Sam	1/2	0	0	0	0	0	0	0	0	0	0	1/2
do	0	0	0	0	1	0	0	0	0	0	0	0
not	0	0	0	0	0	1	0	0	0	0	0	0
like	0	0	0	0	0	0	1	0	0	0	0	0
green	0	0	0	0	0	0	0	1	0	0	0	0
eggs	0	0	0	0	0	0	0	0	1	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0
ham	0	0	0	0	0	0	0	0	0	0	0	1
<s>	2/3	0	1/3	0	0	0	0	0	0	0	0	0
</s>	0	0	0	0	0	0	0	0	0	0	0	0

How to Create Relative Frequency Tables?

P(Unigram) =Freq(Unigram)/N
N=20

I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
3/20	1/10	1/10	1/20	1/20	1/20	1/20	1/20	1/20	1/20	**3/20**	3/20

CORPUS

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

Relative Frequencies

	I	am	Sam	do	not	like	green	eggs	and	ham	<s>	</s>
I	0	2/3	0	1/3	0	0	0	0	0	0	0	0
am	0	0	1/2	0	0	0	0	0	0	0	0	1/2
Sam	1/2	0	0	0	0	0	0	0	0	0	0	1/2
do	0	0	0	0	1	0	0	0	0	0	0	0
not	0	0	0	0	0	1	0	0	0	0	0	0
like	0	0	0	0	0	0	1	0	0	0	0	0
green	0	0	0	0	0	0	0	1	0	0	0	0
eggs	0	0	0	0	0	0	0	0	1	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0
ham	0	0	0	0	0	0	0	0	0	0	0	1
<s>	2/3	0	1/3	0	0	0	0	0	0	0	0	0
</s>	0	0	0	0	0	0	0	0	0	0	0	0

Added by Dr. Liao

How to Compute Probability of an Utterance?

- Given an n-gram language model

P(<s> I am Sam </s>) =

P(<s>) *
P(I|<s>) *
P(am| I) *
P(Sam | am) *
P(</s>| Sam)

This is called the *chain rule of probability* using *the markov assumption*

=3/20*2/3*2/3*½*½

Added by Dr. Liao

Markov Assumption

- Estimate the conditional probability of the next word without looking too far in the past

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

For example, the sentence is "<s> I do not like green eggs and ham </s>".

P(eggs| <s> I do not like green) = P(eggs | green) **Using a bigram model**

P(eggs| <s> I do not like green) = P(eggs | like green) **Using a trigram model**

P(eggs| <s> I do not like green) = P(eggs | not like green) **Using a 4-gram model**

etc …

Added by Dr. Liao

Relative Frequencies use Two Tables

N-gram table

and

(N-1)-gram table

How to Generate Sentences

Added by Dr. Liao

Let's see BeRP example again…

i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
2533	927	2417	746	158	1093	341	278	3000	3000

Unigram table of raw frequencies

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	5	827	0	9	0	0	0	2	0	0
want	2	0	608	1	6	6	5	1	0	0
to	2	0	4	686	2	0	6	211	0	0
eat	0	0	2	0	16	2	42	0	0	34
chinese	1	0	0	0	0	82	1	0	0	23
food	15	0	15	0	1	4	0	0	0	12
lunch	2	0	0	0	0	1	0	0	0	9
spend	1	0	1	0	0	0	0	0	1	17
<start>	45	0	30	0	15	10	3	0	0	0
<end>	0	0	0	0	3	23	6	34	0	0

Bigram table of raw frequencies

Content Added by Dr. Liao

P(I want to eat chinese food)?

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

Relative Frequency Table

How to Generate Sentences

We know to begin the sentences we want to use the <start> tag

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

How to Generate Sentences

Sentences begin with the <start> tag

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0

Now we are only interested in those words that follow <start>
(the non-zero elements)

Why?

Because we are using our language model
(the relative frequency table)
to generate the words in our sentence

How to Choose?

If we pick the one with the highest probability, our sentences are not going to change very much

So

randomly pick one based on the **distribution**

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

Which of the five choices do we choose

Randomly pick one based on its distribution

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

0.015 0.01 0.005 0.003 0.001

To do this we need to normalize <start> out of the distribution

Watch …

Which of the five choices do we choose

Randomly pick one based on its distribution

To do this we need to normalize <start> out of the distribution

Watch …

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

Now
We are going to use this sum to normalize out <start>

$$0.015 + 0.01 + 0.005 + 0.003 + 0.001 = 0.034$$

$$\frac{0.015}{0.034} = 0.44 \qquad \frac{0.01}{0.034} = 0.29 \qquad \frac{0.005}{0.034} = 0.15 \qquad \frac{0.003}{0.034} = 0.09 \qquad \frac{0.001}{0.034} = 0.03$$

This is a divide sign

Which of the five choices do we choose

Randomly pick one based on its distribution

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

$$0.015 + 0.01 + 0.005 + 0.003 + 0.001 = 0.034$$

To do this we need to normalize <start> out of the distribution

$$\frac{0.015}{0.034} = 0.44 \quad \frac{0.01}{0.034} = 0.29 \quad \frac{0.005}{0.034} = 0.15 \quad \frac{0.003}{0.034} = 0.09 \quad \frac{0.001}{0.034} = 0.03$$

Now our probabilities **sum to 1**

Watch …

Which of the **five** choices do we choose

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

$$0.015 + 0.01 + 0.005 + 0.003 + 0.001 = 0.034$$

$$\frac{0.015}{0.034} = 0.44 \qquad \frac{0.01}{0.034} = 0.29 \qquad \frac{0.005}{0.034} = 0.15 \qquad \frac{0.003}{0.034} = 0.09 \qquad \frac{0.001}{0.034} = 0.03$$

These are the probabilities you would pull the words from the bucket

Given a
bucket with only those words that come after <start>
we have probability 0.03 of picking *lunch.*

Which of the five choices do we choose

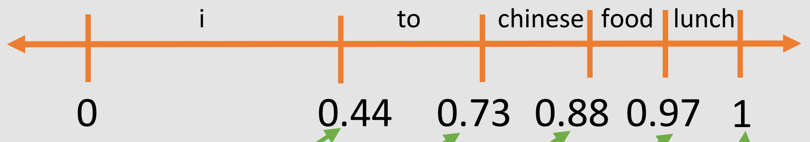
	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

$$0.015 + 0.01 + 0.005 + 0.003 + 0.001 = 0.034$$

$$\frac{0.015}{0.034} = 0.44 \quad \frac{0.01}{0.034} = 0.29 \quad \frac{0.005}{0.034} = 0.15 \quad \frac{0.003}{0.034} = 0.09 \quad \frac{0.001}{0.034} = 0.03$$

	i	to	chinese	food	lunch
<start>	0.44	0.29	0.15	0.09	0.03

We can plot these probabilities a line from 0 to 1



0 + 0.44 + 0.29 + 0.15 + 0.09 + 0.03

Which of the five choices do we choose

To pick a word that follows <start>:
Pick a random number
between 0 and 1,
and then see where it falls on the distribution.

	i	to	chinese	food	lunch
<start>	0.015	0.01	0.005	0.003	0.001

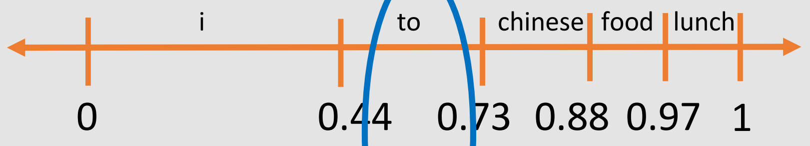
$$0.015 + 0.01 + 0.005 + 0.003 + 0.001 = 0.034$$

$$\frac{0.015}{0.034} = 0.44 \quad \frac{0.01}{0.034} = 0.29 \quad \frac{0.005}{0.034} = 0.15 \quad \frac{0.003}{0.034} = 0.09 \quad \frac{0.001}{0.034} = 0.03$$

	i	to	chinese	food	lunch
<start>	0.44	0.29	0.15	0.09	0.03

So say our random number generator returned the value 0.65,

what is our next word?



=> to

So then we start the process again with ‘to’

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

How to Generate Sentences?

The Shannon Visualization Method

- **Four Steps**
  - Choose a **random bigram** **(<s>, w)** according to its probability
  - Now choose a **random bigram** **(w, x)** according to its probability
  - And so on **until we choose** **</s>**
  - Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Added from Stanford Course materials

Approximating Shakespeare

Unigram

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

Every enter now severally so, let

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

Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

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

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

What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.

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

Indeed the duke; and had a very good friend.

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

Quadrigram

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

Will you not tell me who I am?

It cannot be but so.

Indeed the short and the long. Marry, 'tis a noble Lepidus.

Added from Stanford Course materials

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

Zeros & Sparsity

Added by Dr. Liao

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

  P("offer" | denied the) = 0

- Test set
  … denied the offer
  … denied the loan

Added from Stanford Course materials

Sparsity

As N increases, the **accuracy** of our model increases

But

As N increases, the **sparsity** of our model increases

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

LOOK AT ALL THE ZEROS

Does this mean that P(want|english) = 0?

With the model, yes but in real life?

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0064	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

$$P(I\ want\ to\ eat\ English\ Food) =$$

$P(i|<start>)$ *
$P(want|i)$ *
$P(to\ |\ want)$ *
$P(eat|to)$ *
$P(english|eat)$ *
$P(food|english)$ *
$P(<end>|food) = ?$

$P(i|<start>) = 0.015$
$P(want|i) = 0.33$
$P(to\ |\ want) = 0.66$
$P(eat|to) = 0.28$
$P(english|eat) = 0$
$P(food|english) = 0$
$P(<end>|food) = 0.004$

Uh oh!

$$P(I\ want\ to\ eat\ English\ Food) = 0\ ?$$

Sparsity & Recap MLE

- Sparsity is a major problem for Maximum Likelihood Estimation (MLE)
- This is MLE => $P(w_1^n) = \prod_{k=1}^{n} P(w_k|w_{k-1})$
- MLE with example:

$$P(the\ magical\ unicorn) = \\ P(the) * \\ P(magical|the) * \\ P(unicorn|magical)$$

These probabilities are referred to as **Relative Frequency**

Relative Frequency

- Sparsity is a major problem for Maximum Likelihood Estimation (MLE)
- This is MLE => $P(w_1^n) = \prod_{k=1}^{n} P(w_k | w_{k-1})$
-

This is **Relative Frequency** => $P(w_k | w_{k-1})$

$$P(unicorn|magical) = \frac{Frequency(magical\ unicorn)}{Frequency(magical)}$$

Sparsity

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	0.33	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0064	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

Because we don't see "<start> eat" in the text does this mean it doesn't occur ever?

Is P(*<start> eat)* really zero?

N-gram Smoothing

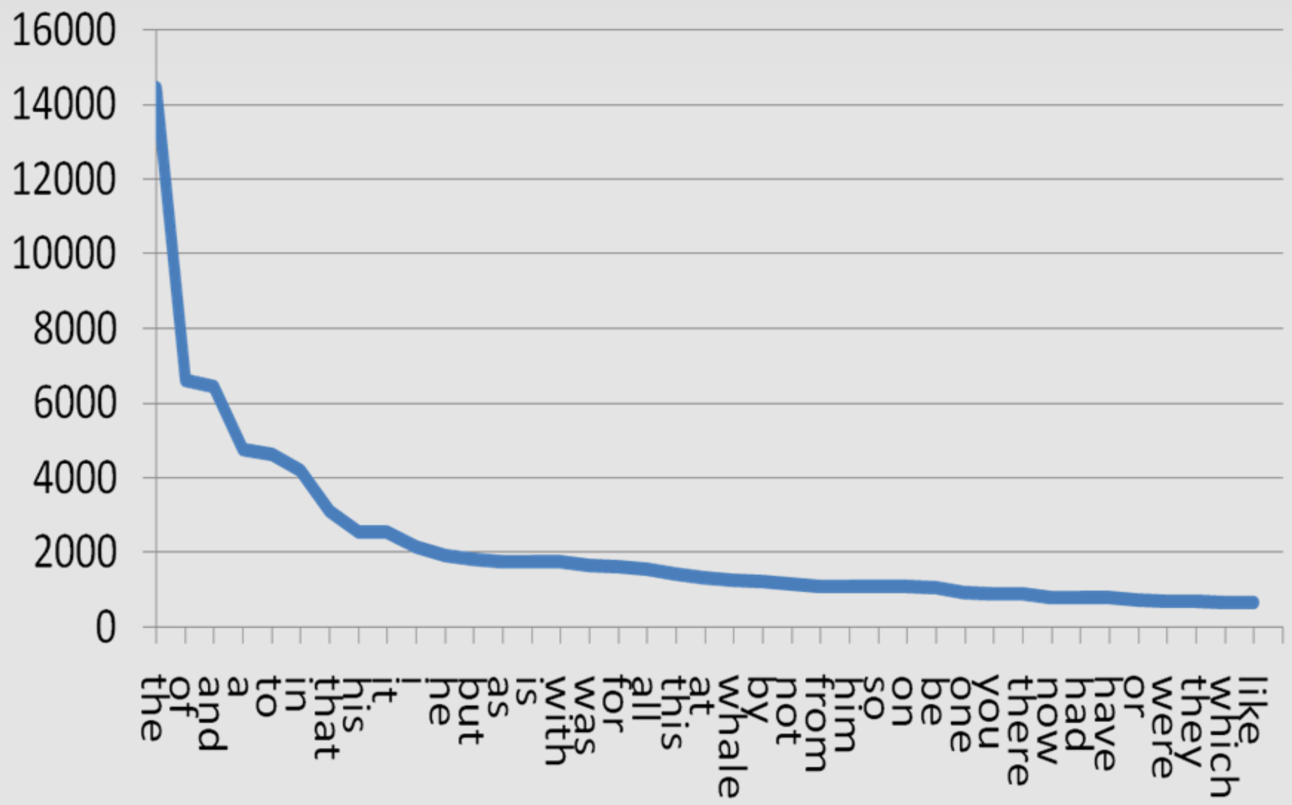
Added by Dr. Liao

Smoothing

- Exploit the Zipfian distribution of words
- Two smoothing methods:
  - **Laplace Smoothing**
  - **Good-Turning**
- The basic idea is that we take a little from everything we see and give it to what we don't see
- Robin Hood: stealing from the rich and giving to the poor

Zipfian Distribution

- Words follow a Zipfian Distribution
  - Small number of words occur very frequently
  - A large number of words are only seen once
- Zipf's Law
  - A word's frequency is approximately inversely proportional to its rank in the word distribution list
- Great Video on Zipf's Law
  - https://www.youtube.com/watch?v=fCn8zs912OE



Integrated by Dr. Liao

Laplace Smoothing (Add-1 smoothing)

- Simple metric : adds **one** to each count
- Pretend we saw each word one more time than we did

$$P(w_i) = \frac{frequency(w_i)}{N} \qquad P_{Laplace}(w_i) = \frac{frequency(w_i) + \mathbf{1}}{N + \boldsymbol{V}}$$

N = the number of tokens in our corpus
V = the number of types in our corpus

Adding V because you've added one to each w seen in your corpus

Contents added by Dr. Liao

Discounted Frequencies

$$P_{Laplace}(w_i) = \frac{\boldsymbol{frequency(w_i)} + \mathbf{1}}{N + \boldsymbol{V}}$$

$$P_{Laplace}(w_i) = \frac{\boldsymbol{frequency}^*(w_i)}{\boldsymbol{N}}$$

$$\boldsymbol{frequency}^*(w_i) = (\boldsymbol{frequency(w_i)} + \mathbf{1})\frac{\boldsymbol{N}}{N + \boldsymbol{V}}$$

N = the number of tokens in our corpus
V = the number of types in our corpus

Discounted Frequencies and Probabilities

$$frequency^*(w_i) = (frequency(w_i) + 1)\frac{N}{N+V}$$

$$P_{Laplace}(w_i) = \frac{frequency^*(w_i)}{N}$$

1: $P_{Laplace}(w_i) = \frac{(frequency(w_i) + 1)\frac{N}{N+V}}{N}$

3: $P_{Laplace}(w_i) = \frac{N(frequency(w_i)+1)}{N+V} * \frac{1}{N}$

2: $P_{Laplace}(w_i) = \frac{\frac{N(frequency(w_i)+1)}{N+V}}{N}$

4: $P_{Laplace}(w_i) = \frac{(frequency(w_i) + 1)}{N+V}$

Laplace Smoothing on Conditional Probabilities

$$P(w_i) = \frac{frequency(w_i)}{N} \Rightarrow P_{Laplace}(w_i) = \frac{frequency(w_i) + \mathbf{1}}{N + \boldsymbol{V}}$$

$$P(w1|w2) = \frac{frequency(w_1 w_2)}{frequency(w_1)} \Rightarrow P_{Laplace}(w_i|wi\ \_\ {}_1) = ?$$

$$P_{Laplace}(w_n|w_{n-1}) = \frac{frequency(w_{n-1}w_n) + \mathbf{1}}{frequency(w_{n-1}) + V}$$

V = the number of types in our corpus

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	5	827	0	9	0	0	0	2	0	0
want	2	0	608	1	6	6	5	1	0	0
to	2	0	4	686	2	0	6	211	0	0
eat	0	0	2	0	16	2	42	0	0	34
chinese	1	0	0	0	0	82	1	0	0	23
food	15	0	15	0	1	4	0	0	0	12
lunch	2	0	0	0	0	1	0	0	0	9
spend	1	0	1	0	0	0	0	0	1	17
<start>	45	0	30	0	15	10	3	0	0	0
<end>	0	0	0	0	3	23	6	34	0	0

i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
2533	927	2417	746	158	1093	341	278	3000	3000

$$P_{Laplace}(w_n|w_{n-1}) = \frac{frequency(w_{n-1}w_n) + 1}{frequency(w_{n-1}) + V}$$

V = 1446

$$P_{Laplace}(want|i) = \frac{frequency(i\ want) + 1}{frequency(i) + \boldsymbol{V}}$$

$$= \frac{827+1}{2533+1446} = 0.21$$

	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

Integrated and added by Dr. Liao

How smoothing affects probabilities?

	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

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	0.002	**0.33**	0	0.0036	0	0	0	0.00079	0	0
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0064	0.0011	0	0
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087	0	0
eat	0	0	0.0027	0	0.021	0.0027	0.056	0	0	0.011
chinese	0.0063	0	0	0	0	0.52	0.0063	0	0	0.008
food	0.014	0	0.014	0	0.00092	0.0037	0	0	0	0.004
lunch	0.0059	0	0	0	0	0.0029	0	0	0	0.003
spend	0.0036	0	0.0036	0	0	0	0	0	1	0.006
<start>	0.015	0	0.01	0	0.005	0.003	0.001	0	0	0
<end>	0	0	0	0	0.001	0.007	0.002	0.011	0	0

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.

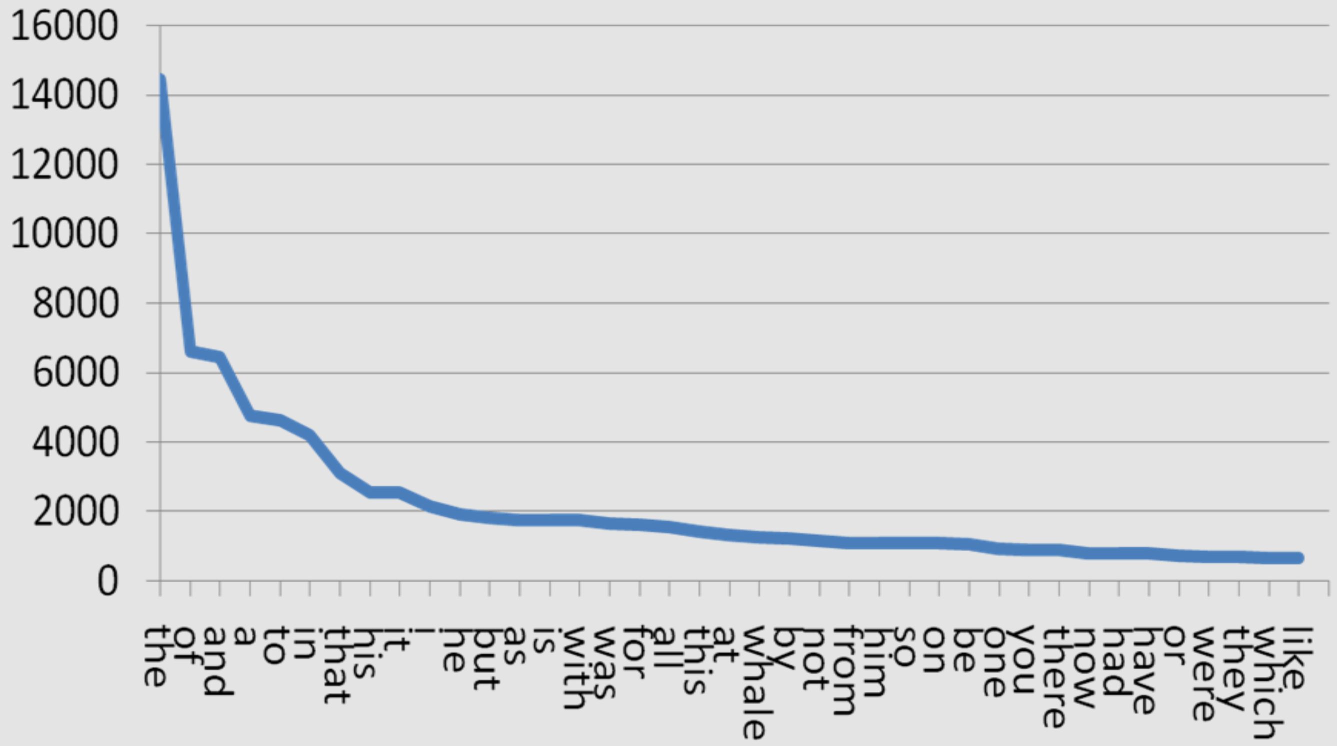
Added from Stanford Course materials

Good-Turing

- Add-1 smoothing (Laplace smoothing) is a bit brute force
- Few more elegant ways to smooth
  - **Good-Turning**
  - Witten-Bell
  - Kneser-Ney

Good-Turing

- Intuition
  - Use the count of things you have seen *once* to help estimate the count of things you've *never seen*



Good-Turing

Based on computing N_c which is the number of N-grams that occur **c** times

frequency of frequency

$$N_o = \#\ of\ bigrams\ with\ count\ 0$$
$$N_1 = \#\ of\ bigrams\ with\ count\ 1$$

...

$$N_c = \#\ of\ bigrams\ with\ count\ c$$

Adjust frequencies

$$frequency^{*}(w_i) = (frequency(w_i) + 1)\frac{N}{N+V}$$

Laplace Smoothing

$$\boldsymbol{frequency}^{*}(w_i) = (frequency(w_i) + 1)\frac{\boldsymbol{N_{c+1}}}{\boldsymbol{N_c}}$$

Good Turing Smoothing

$$P_{smoothing}(w_n|w_{n-1}) = \frac{\boldsymbol{frequency}^{*}(w_{n-1}w_n)}{\boldsymbol{frequency}^{*}(w_{n-1})}$$

But what about unseen bigrams?

$$P_{gt}(unseen) = \frac{N_1}{N_o}$$

N_1 = number of bigrams seen 1 time
N_o = total number of bigrams in the corpus

Example

Frequency	Frequency(Frequency)
0	2081496
1	5315
2	**1419**
3	**642**
4	381
5	311
6	196
Frequency	Frequency(Frequency)
…	…
2533	**2**
2534	**2**
….	….
M	1

Unigram

i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
2533	927	2417	746	158	1093	341	278	3000	3000

Bigram

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	5	827	0	9	0	0	0	**2**	0	0

i spend occurs **twice** in our corpus

$$frequency^*(w_i) = (frequency(w_i) + 1)\frac{N_{c+1}}{N_c}$$

$$frequency^*(i\ spend) = (\boldsymbol{frequency(i\ spend)} + 1)\frac{\boldsymbol{N_3}}{\boldsymbol{N_2}}$$

$$frequency^*(i\ spend) = (\boldsymbol{2} + 1)\frac{\boldsymbol{642}}{\boldsymbol{1419}} = 1.36$$

$$\boxed{P_{gt}(spend \mid i) = \frac{frequency^*(i\ spend)}{frequency^*(i)} = \frac{1.36}{\boldsymbol{2534}} = 0.00054}$$

How do we know this?

$$frequency^*(i) = (\boldsymbol{frequency(i)} + 1)\frac{\boldsymbol{N_{2534}}}{\boldsymbol{N_{2533}}} = (\boldsymbol{2533} + 1)\frac{\boldsymbol{2}}{\boldsymbol{2}} = \boldsymbol{2534}$$

Revised by Dr. Liao

Frequency	Frequency(Frequency)
0	2081496
1	5315
2	1419
3	642
4	381
5	311
6	196
Frequency	Frequency(Frequency)
…	…
2533	2
2534	2
….	….
M	1

	i	want	to	eat	chinese	food	lunch	spend	<start>	<end>
i	5	827	0	9	0	0	0	2	0	0

i spend occurs twice in our corpus

$$frequency^*(w_i) = (frequency(w_i) + 1)\frac{N_{c+1}}{N_c}$$

$$frequency^*(i\ spend) = (frequency(i\ spend) + 1)\frac{N_3}{N_2}$$

$$frequency^*(i\ spend) = (2 + 1)\frac{642}{1419} = 1.36$$

$$\boxed{P_{gt}(spend \mid i) = \frac{frequency^*(i\ spend)}{frequency^*(i)} = \frac{1.36}{2534} = 0.00054}$$

$$P^*(\boldsymbol{i\ to}) = \frac{N_1}{N_0} = \frac{5315}{2081496} = 0.003$$

Unseen

Contents added by Dr. Liao

What happens when $N_{c+1} = 0$

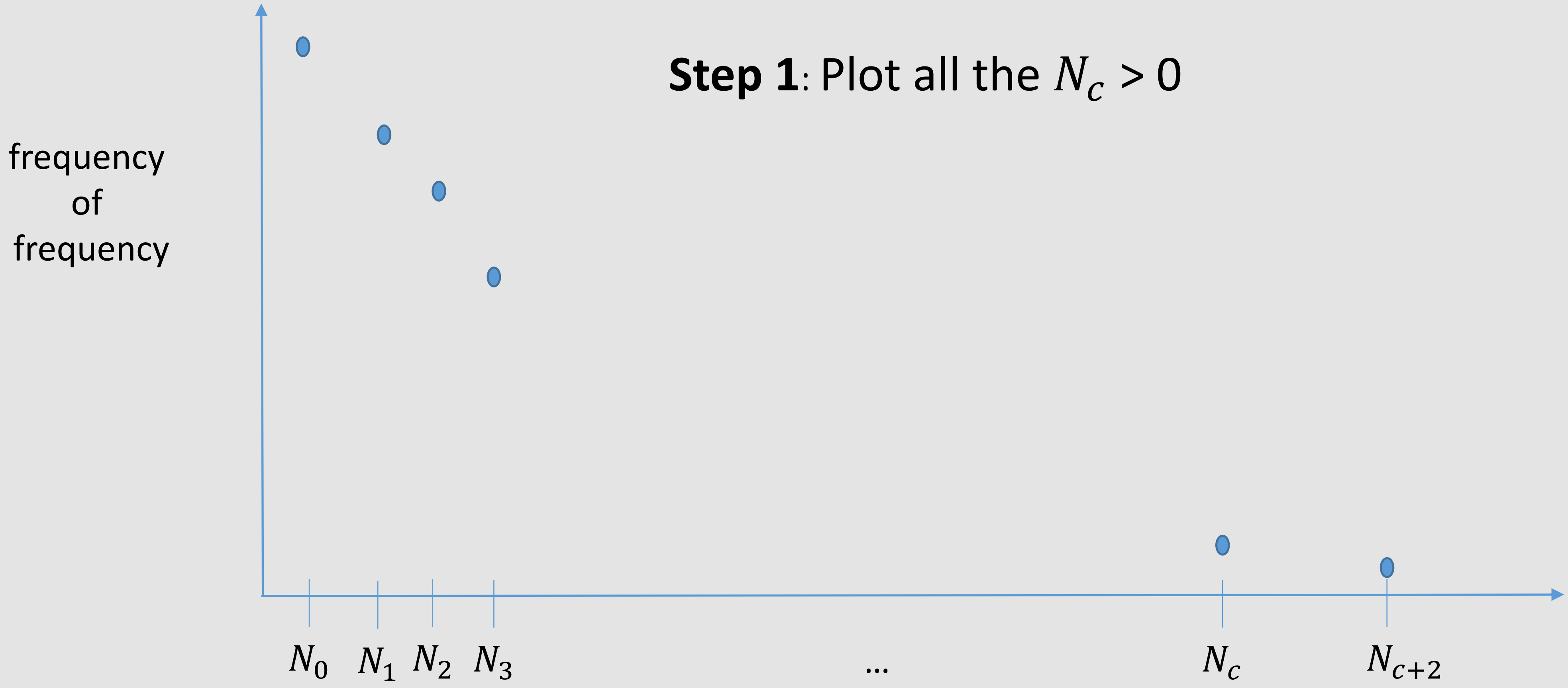
Frequency	Frequency(Frequency)
0	2081496
1	5315
2	1419
3	642
4	381
5	311
6	196
Frequency	Frequency(Frequency)
…	…
2533	2
2534	2
2535	0
….	….

$$frequency^*(w_i) = (frequency(w_i) + 1)\frac{\boldsymbol{N_{c+1}}}{N_c}$$

Simplest thing is to perform linear regressions and replace the value of N_{c+1} with regression value whenever N_{c+1} = 0

Estimating when $N_{c+1} = 0$

Frequency of frequency
Is the number of n-grams
That occurred N_{c+1} times

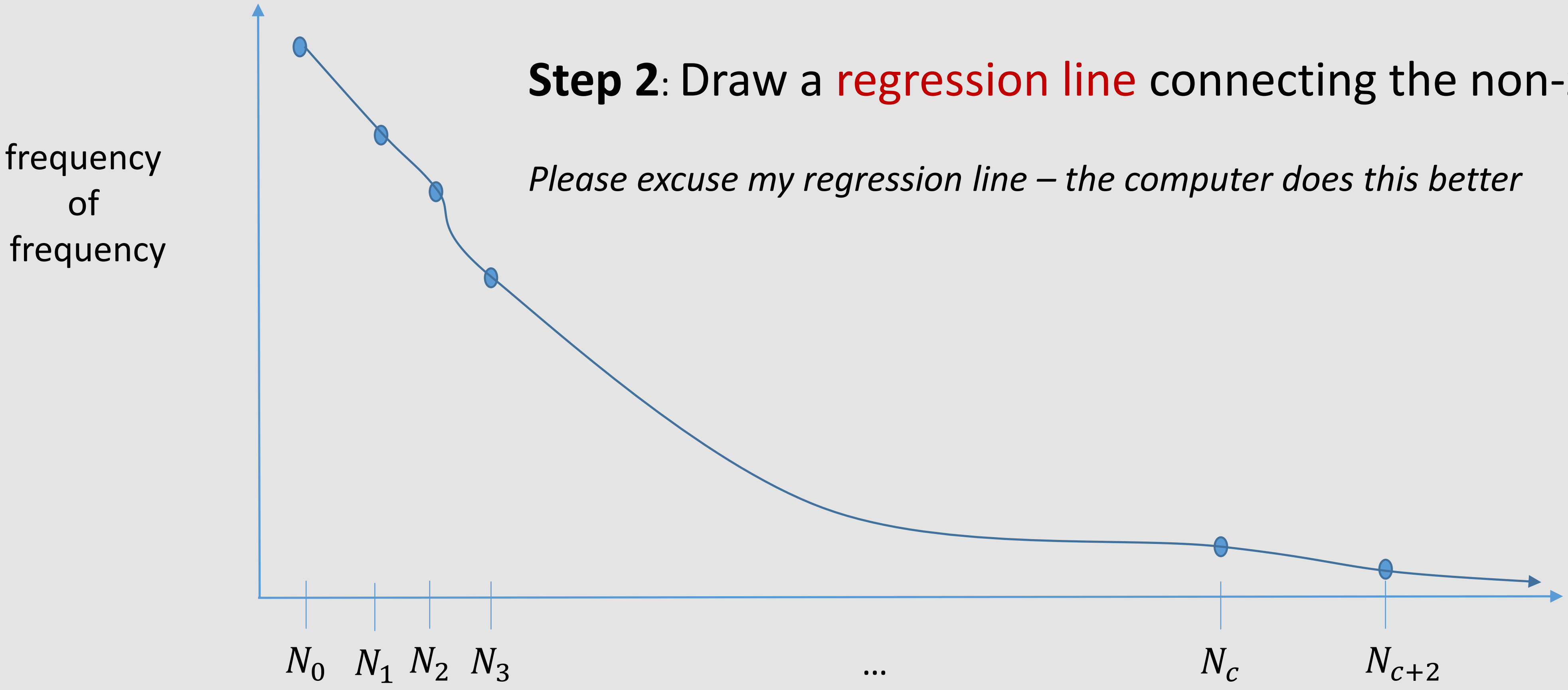


Estimating when $N_{c+1} = 0$

Frequency of frequency
Is the number of n-grams
That occurred N_{c+1} times

Step 2: Draw a regression line connecting the non-zero points

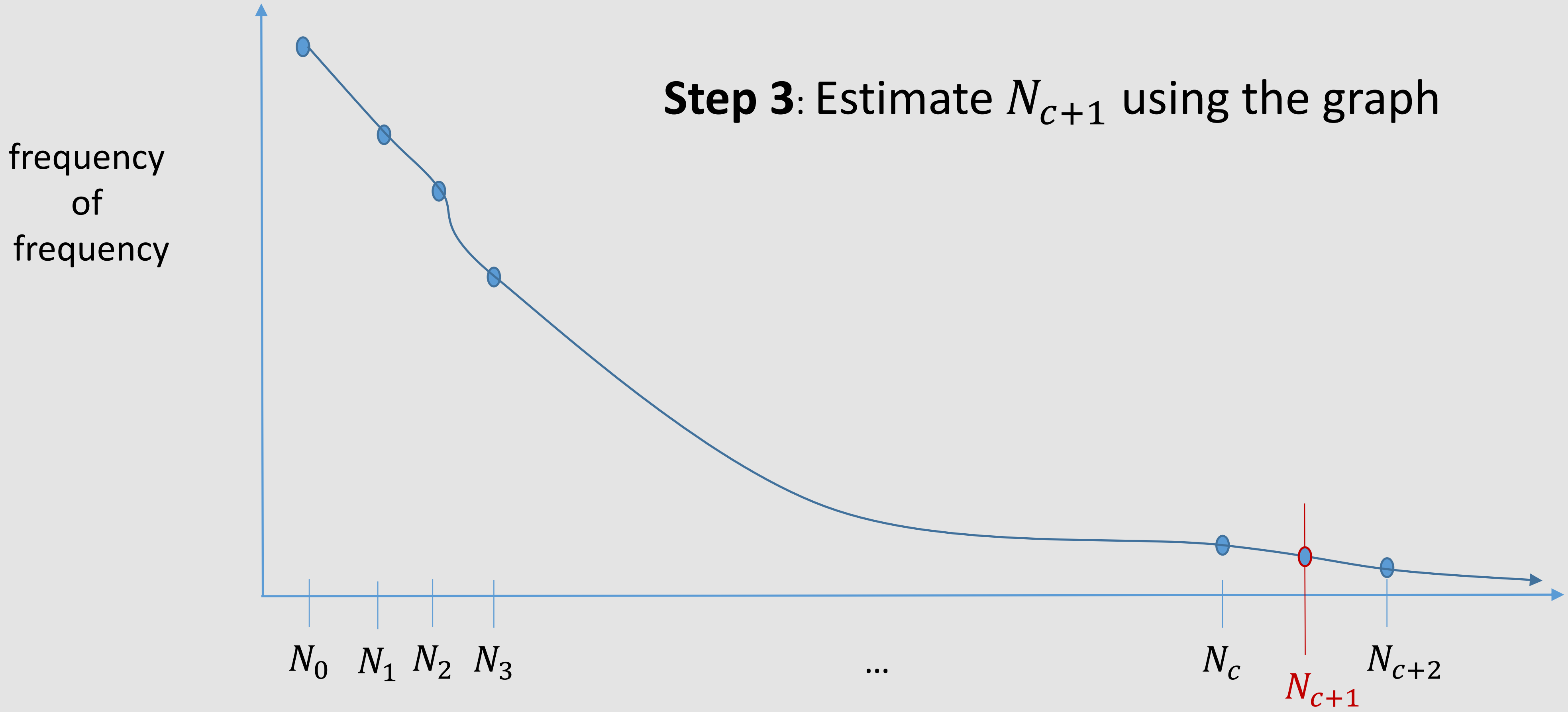
Please excuse my regression line – the computer does this better



Estimating when N_{c+1} = 0

Frequency of frequency
Is the number of n-grams
That occurred N_{c+1} times

Step 3: Estimate N_{c+1} using the graph

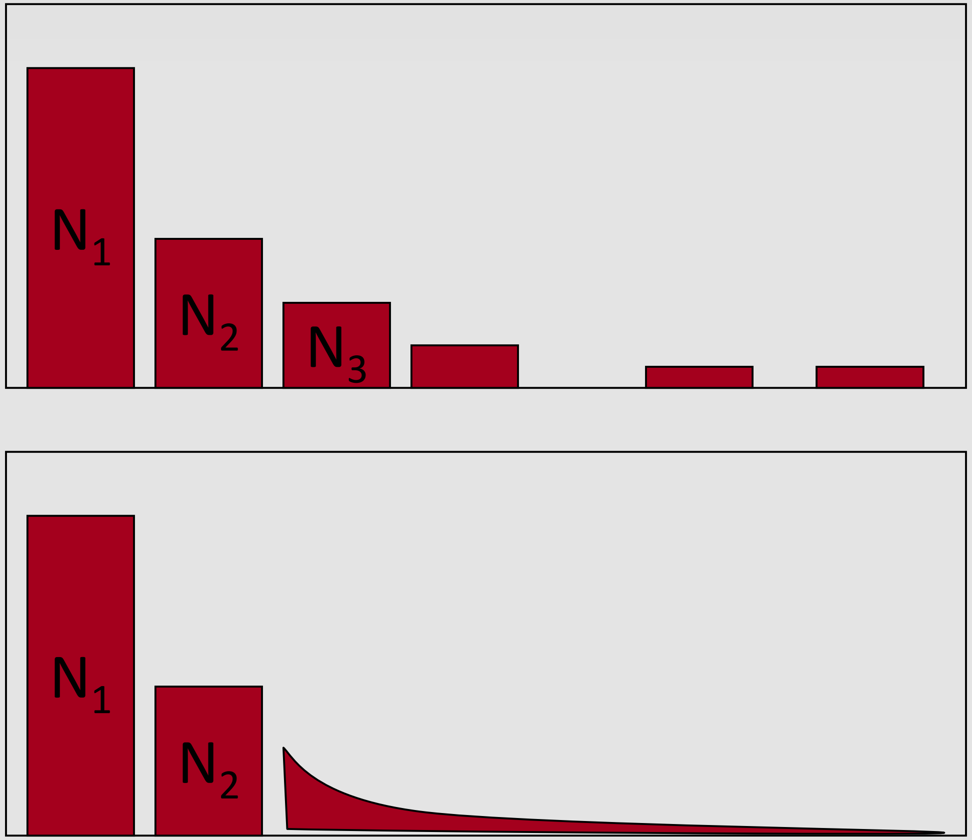


Good-Turing Complications

(slide from Dan Klein)

- Problem: what about "the"? (say c=4417)
  - For small k, $N_k > N_{k+1}$
  - For large k, too jumpy, zeros wreck estimates

  - Simple Good-Turing [Gale and Sampson]: replace empirical N_k with a best-fit power law once counts get unreliable



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
  - 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)

Added from Stanford Course materials

Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

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

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

Added from Stanford Course materials

N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization/classfication, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid Backoff
    - Works **well** in practice

Added from Stanford Course materials

Next up

- Next time:
  - POS tagging (read Chapter 5)
  - Student presentations