Statistical Estimators

MSc Prep Course 2011

Liling, Noushin, Sevtap, Wei

Today’s Menu

- N-grams
- Maximum Likelihood Estimators
- Laplace’s Law, Lidstone’s Law, Jeffreys-Perks Law
- Held out Estimation
- Cross Validation
- Good-Turing Estimation

Maximum Likelihood Estimation

What is an N-gram?

- Question: Please turn your homework ...
- what is the most likely word to finish this sentence?
- what is an unlikely word to finish this sentence?

N-gram

- informally, a way to model that human intuition
- formally, word prediction with a probabilistic model

N-gram

- Question: why is it called N-gram?
- Predicts the last word from the previous N-1 words.

N-gram

- 1-gram: 1-token sequence of words
- 2-gram: 2-token sequence of words
- 3-gram: 3-token sequence of words
- ...
- N-gram: N-token sequence of words

N-gram

- Exercise: List some of the 1-grams, 2-grams, 11-grams in the following text
- Bir varmis bir yokmus. Evvel zaman icinde kalbur zaman icinde.

N-gram

- Q: Why does N-gram prediction work?
- It turns out, computing the probability of the next word is closely related to computing the probability of a sequence of words

N-gram

- Q: Which sentence is more likely to appear in a text
- 1. ...all of a sudden I notice three guys standing on the sidewalk...
- 2. ...on guys all I notice sidewalk three sudden standing the...

N-gram

- N-gram is an N-token sequence of words
- Can also mean, the model which uses N-grams to predict the next word or an entire sequence of N words.

N-gram

- An important tool in speech and language processing
- Handwriting recognition
  - P("I have a gun") >> P("I have a gub")
- Machine translation
  - P("briefed reporters") >> P("briefed to reporters")

N-gram

- Spelling correction
  - P(“in about fifteen minutes”) >> P(“in about fifteen minuets”)
- Augmentative communication
  - P(“in” | ”turn your homework”) >> P(“the” | ”turn your homework”)

N-gram (estimating the next word)

- P(w|h) read as probability of a word w given some history h
- Ex: P(the | its water is so transparent that)
- Q: How can we compute this probability?

N-gram (estimating the next word)

- One way of computing it:
  - estimate it from the relative frequency counts.
  - but how?

N-gram (estimating the next word)

- Step 1: take a large corpus
- .... and he saw the lake .... its water is so transparent that he .... and its water is so transparent that she ... and its water is so transparent that the ...

N-gram (estimating the next word)

- Step 2: count the number of times “its water is so transparent that” occurs in the corpus, call this number count1
- Step 3: count the number of times “its water is so transparent that the” occurs in the corpus, call this number count2
- Step 4: Divide count2 by count1

N-gram (estimating the next word)

- P(the | its water is so transparent that) = Count(its water is so transparent that the) / Count(its water is so transparent that)
- works fine in many cases
- not enough data to make it work in most cases.

N-gram (estimating the sequence of words)

- What about P(its water is so transparent)?
- Looks like a joint probability of entire sequence of words
- Q: How can we compute this probability?

N-gram (estimating the sequence of words)

- One way of computing it:
  - estimate it from the relative frequency counts.
  - but how?

N-gram (estimating the sequence of words)

- Step 1: take a large corpus
- ... and he went there to fetch some ... its water is so transparent ... again said he its water is so transparent as to ...

N-gram (estimating the sequence of words)

- Step 2: count the number of times “its water is so transparent” occurs in the corpus, call this number count1
- Step 3: count the number of all 5-grams in the corpus, call this number count2
- Step 4: Divide count1 by count2

N-gram(estimation)

- We did a lot of counting
- Q: Isn’t there a cleverer way of estimating the probabilities for the next word and for the entire sequence of words?

N-gram (some notation)

- $P(X_1=\text{"the"})$ read as probability of random variable X_1 taking the value "the" aka $P(\text{the})$
- $w_1 \ldots w_n$ see as sequence of n words
- $P(X=w_1, Y=w_2, Z=w_3, \ldots, W=w_n)$ see as the joint probability of each word in a sequence having a particular value, aka $P(w_1, w_2, \ldots, w_n)$

Computing $P(w_1, w_2, \ldots, w_n)$

- Q: How can we compute the probabilities of entire sequences of words?
- In other words, how can we compute $P(w_1, w_2, \ldots, w_n)$
- Ex: P(its, water, is, so, transparent)

Compt. $P(w_1, w_2, ..., w_n)$

- One way of computing this probability is to use the chain rule
- $P(w_1, w_2, ..., w_n) = P(w_1)\ P(w_2|w_1)\ P(w_3|w_1w_2)\ ...\ P(w_n|w_1w_2...w_{n-1})$
- This formula shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words

Compt. $P(w_1, w_2, ..., w_n)$

- P(it is raining outside) = P(it) P(is|it) P(raining|it is) P(outside|it is raining)
- The red terms are conditional probabilities and yet for long sequences of words, we don't know a way to compute them :(

Compt. $P(w_n|w_1 w_2 \ldots w_{n-1})$

- Q: Can't we use the relative frequency to compute that?
- NO
- that means doing
  - Count(every word followed by every long string)
  - Count(every long string)

Compt. $P(w_n|w_1 w_2 \ldots w_{n-1})$ using N-gram model

- Instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.
- Q: Can we do this?
- Yes, if we assume that Markov is right.

Compt. $P(w_n|w_1 w_2 ... w_{n-1})$ using N-gram model

- Q: What did Markov say?
- The probability of a word depends only on the previous word.
- So P(outside|it is raining) is approximately same as P(outside|raining)

Compt. $P(w_n|w_1 w_2 \dots w_{n-1})$ using N-gram model

- A 2-gram model approximates the conditional probability P(w|h) by just looking one word into the past
- A 3-gram model does that with two words
- An N-gram model does that with N-1 words

Compt. $P(w_n|w_1 w_2 \dots w_{n-1})$ N-gram approximation

- $P(w_n|w_1 w_2 \dots w_{n-1}) =_{approx} P(w_n|w_{n-N+1} \dots w_{n-1})$
- N=1: $P(w_n|w_1 \dots w_{n-1}) =_{approx} P(w_n)$
- N=2: $P(w_n|w_1 \dots w_{n-1}) =_{approx} P(w_n| w_{n-1})$
- N=3: $P(w_n|w_1 \dots w_{n-1}) =_{approx} P(w_n|w_{n-2}\ w_{n-1})$
- The term in red is called the N-gram probability

Computing N-gram probability

- Good that we have an approximation
- Q: how can we estimate an N-gram probability?
- A: the simplest and the most intuitive way is to use Maximum Likelihood Estimation (MLE)

Computing N-gram probability

- To estimate the value of $P(w_n|w_1 w_2 \ldots w_{n-N+1})$ use counts from a corpus and normalize them.
- Normalize: divide a count by some total count so that resulting probabilities fall legally between 0 and 1.

Computing N-gram probability

- Ex: to compute a particular 2-gram probability P(y|x)
- 1. Count how many times this 2-gram occurs in the corpus. Count(xy). count1
- 2. Count all the 2-grams that share the same first word x. Count(x_). count2
- 3. Normalize by dividing count1 by count2

Computing N-gram probability

- $P(w_n | w_1 \ldots w_{n-1}) = \text{Count}(w_{n-N+1} \ldots w_n) / \text{Count}(w_{n-N+1} \ldots w_{n-1})$
- Example in the Speech and Language Processing book, page 123

Computing N-gram probability

- We estimate the N-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix
- This ratio is called relative frequency

Computing N-gram probability

- The use of relative frequencies as a way to estimate probabilities is an example of MLE
- In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M. P(T|M)

Computing N-gram probability

- Example in the Speech and Language Processing book, page 125

References

- Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition. Prentice-Hall.

Laplace’s Law, Lidstone’s Law, Jeffreys-Perks Law

Outline

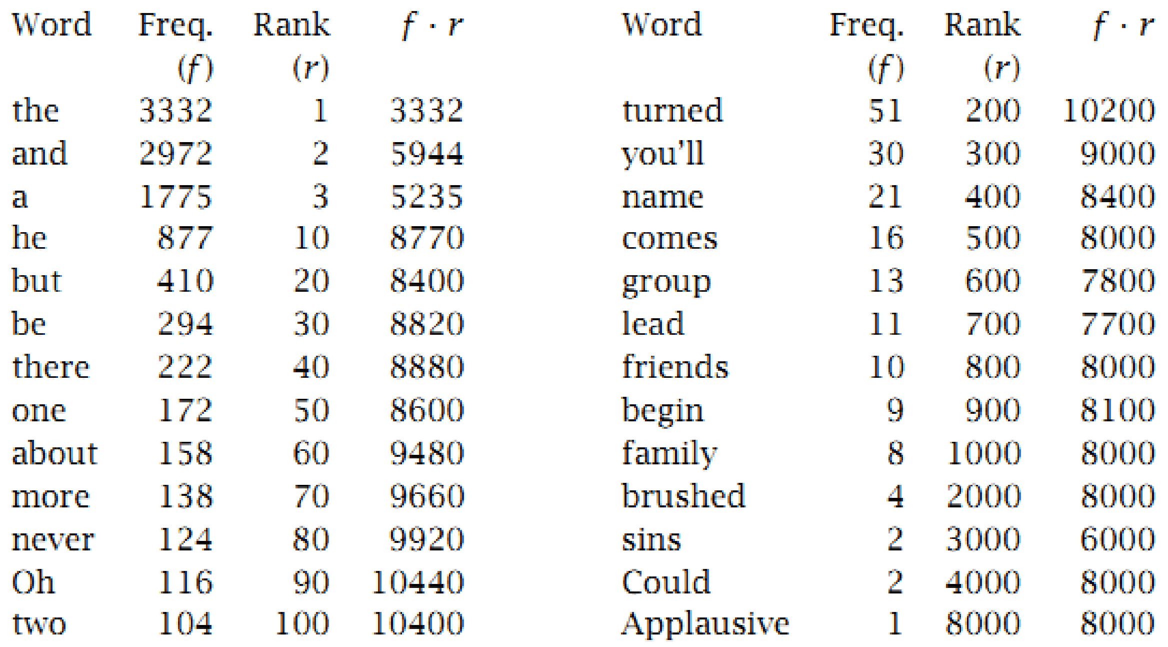
- Motivation: why MLE doesn’t work well?
- Laplace’s Law
- Lidstone’s Law
- Jeffreys-perks Law

Why MLE fails

- Maximize the probability of observed data: assign 0 to unobserved data.
  - Can we enlarge the corpus to avoid this?

- To some extent… Google 5-gram corpus
- According to Zipfs' law:
  - we'll always have rare words not matter how large the corpora. (Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table.)
  - Vocabulary= 20,000, then possible bigrams = 20,000*20,000

Zipfs law



Word	Freq. (f)	Rank (r)	$f \cdot r$
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
about	158	60	9480
more	138	70	9660
never	124	80	9920
Oh	116	90	10440
two	104	100	10400
turned	51	200	10200
you'll	30	300	9000
name	21	400	8400
comes	16	500	8000
group	13	600	7800
lead	11	700	7700
friends	10	800	8000
begin	9	900	8100
family	8	1000	8000
brushed	4	2000	8000
sins	2	3000	6000
Could	2	4000	8000
Applausive	1	8000	8000

Empirical evaluation of Zipf's law on Tom Sawyer.
-- foundations of statistical NLP P24

Example

– Corpus: five Jane Austen novels

– N = 617,091 words

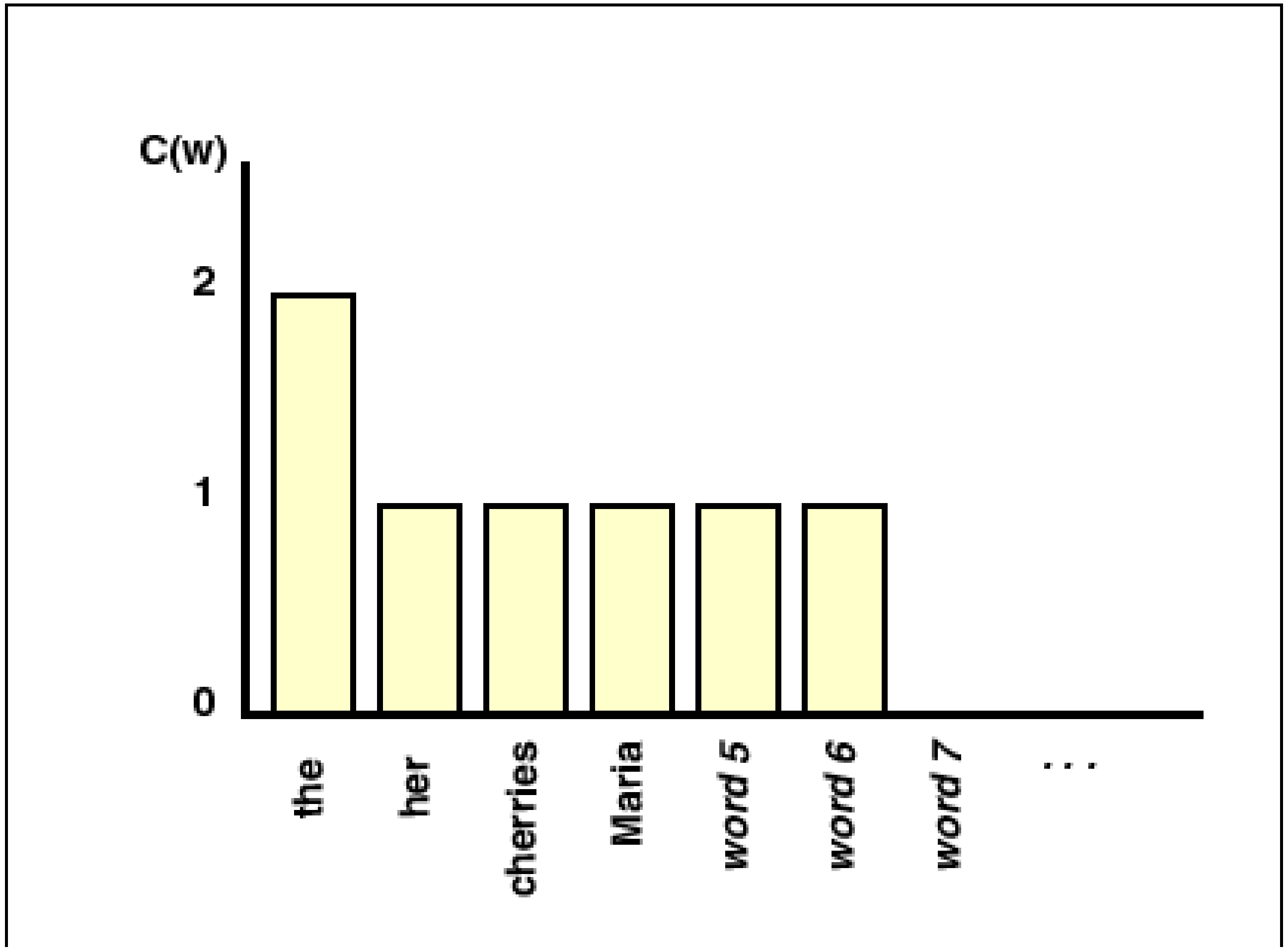
– V = 14,585 unique words

– Task: predict the next word of the trigram "inferior to ________"

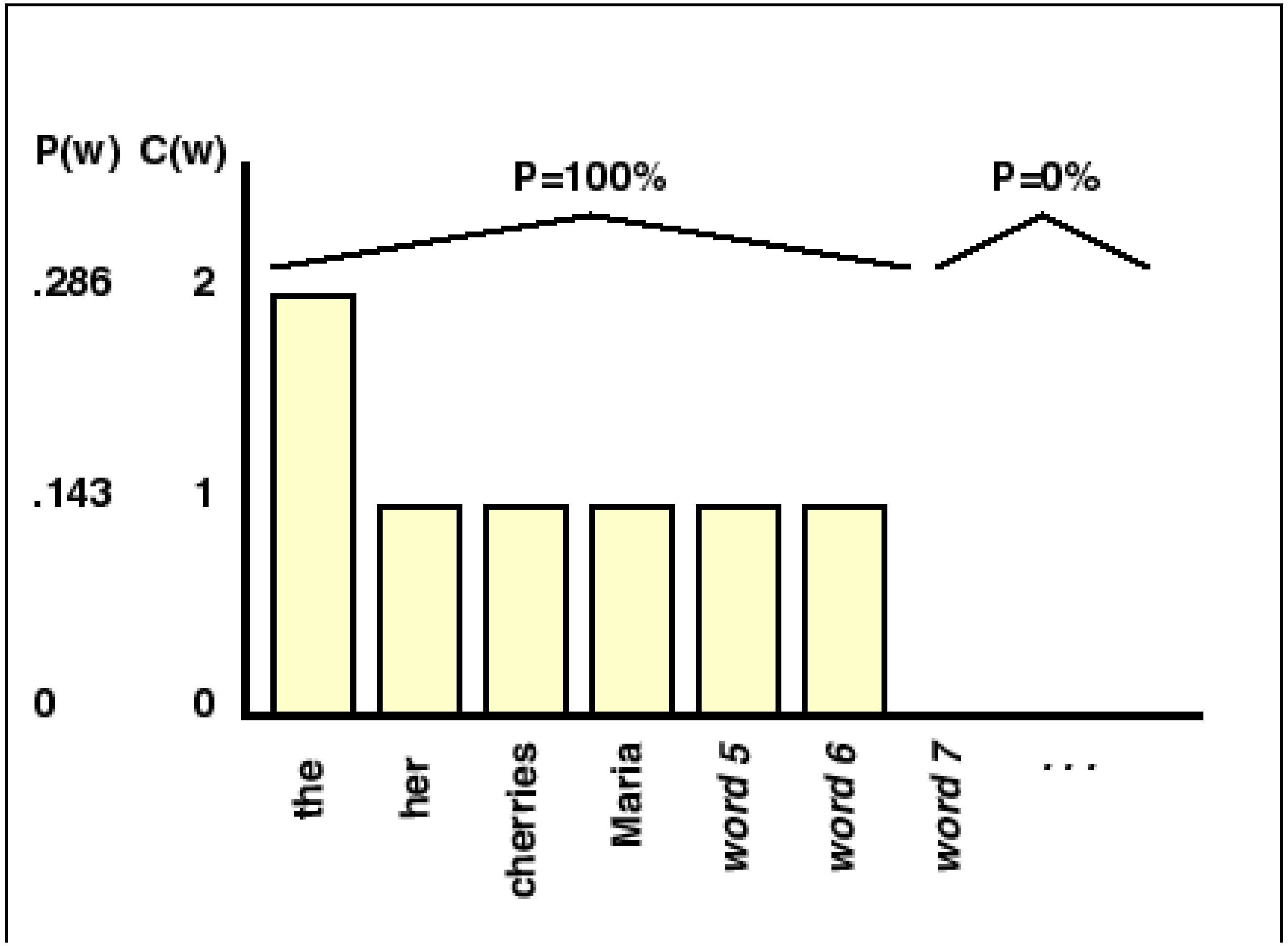
  - from test data, *Persuasion*: "[In person, she was] inferior to *both* [sisters.]"

Instances in the Training Corpus:

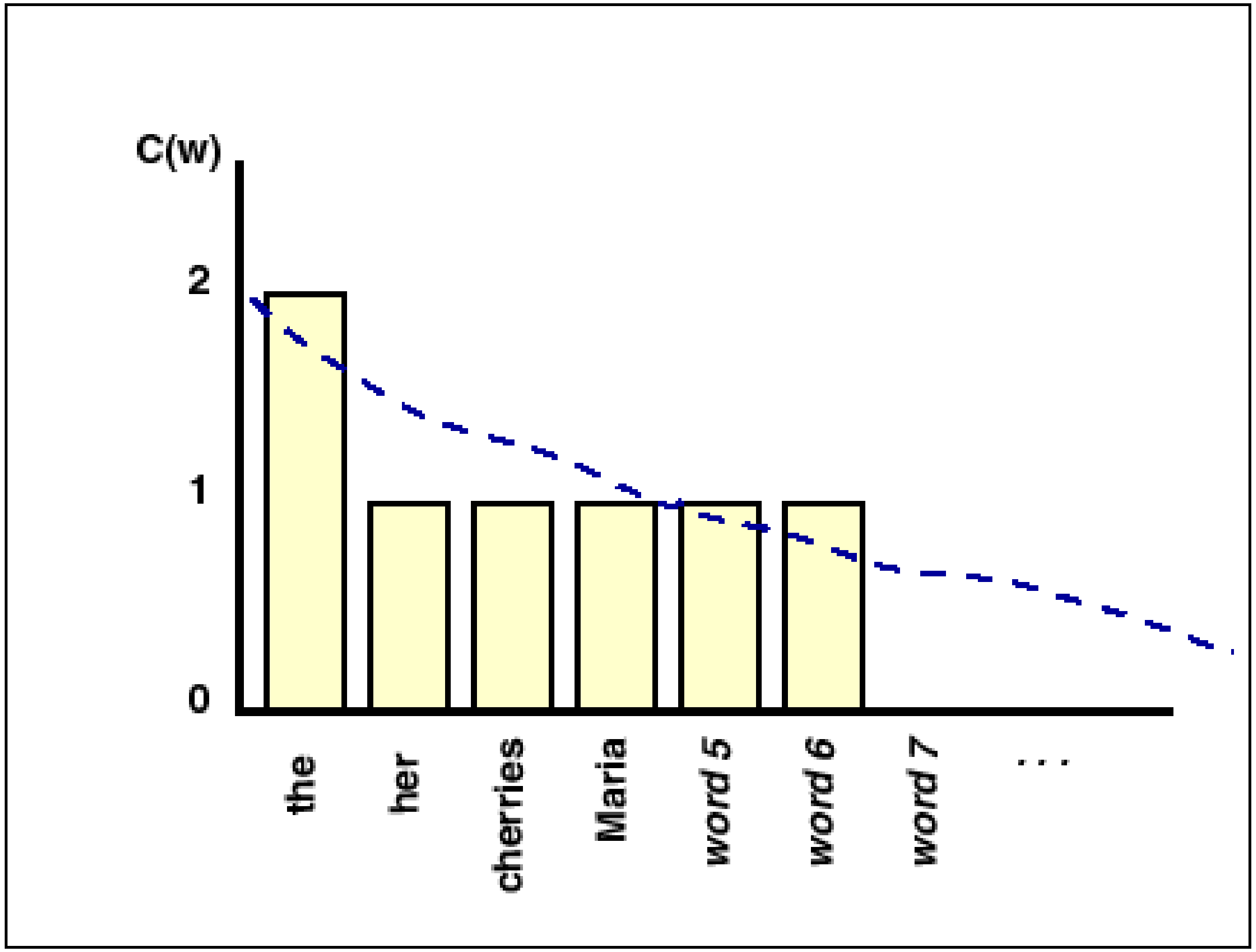
"inferior to __________"



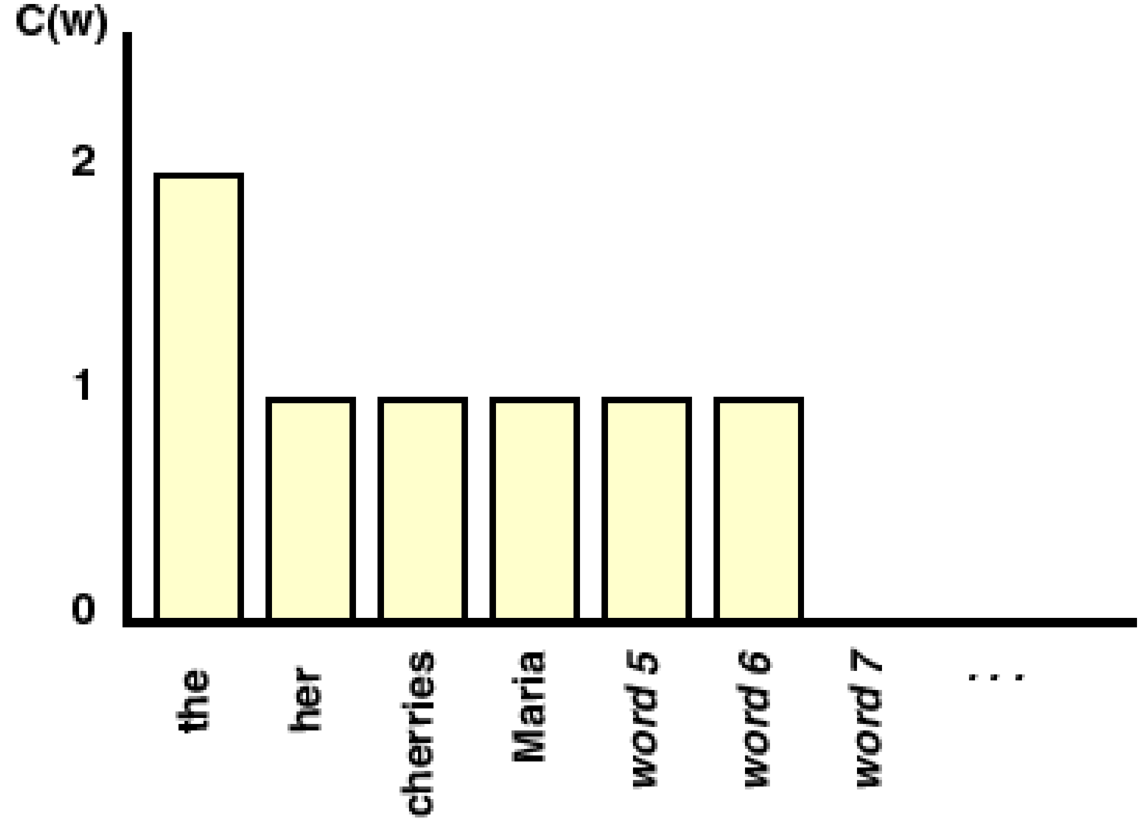
Maximum Likelihood Estimate:



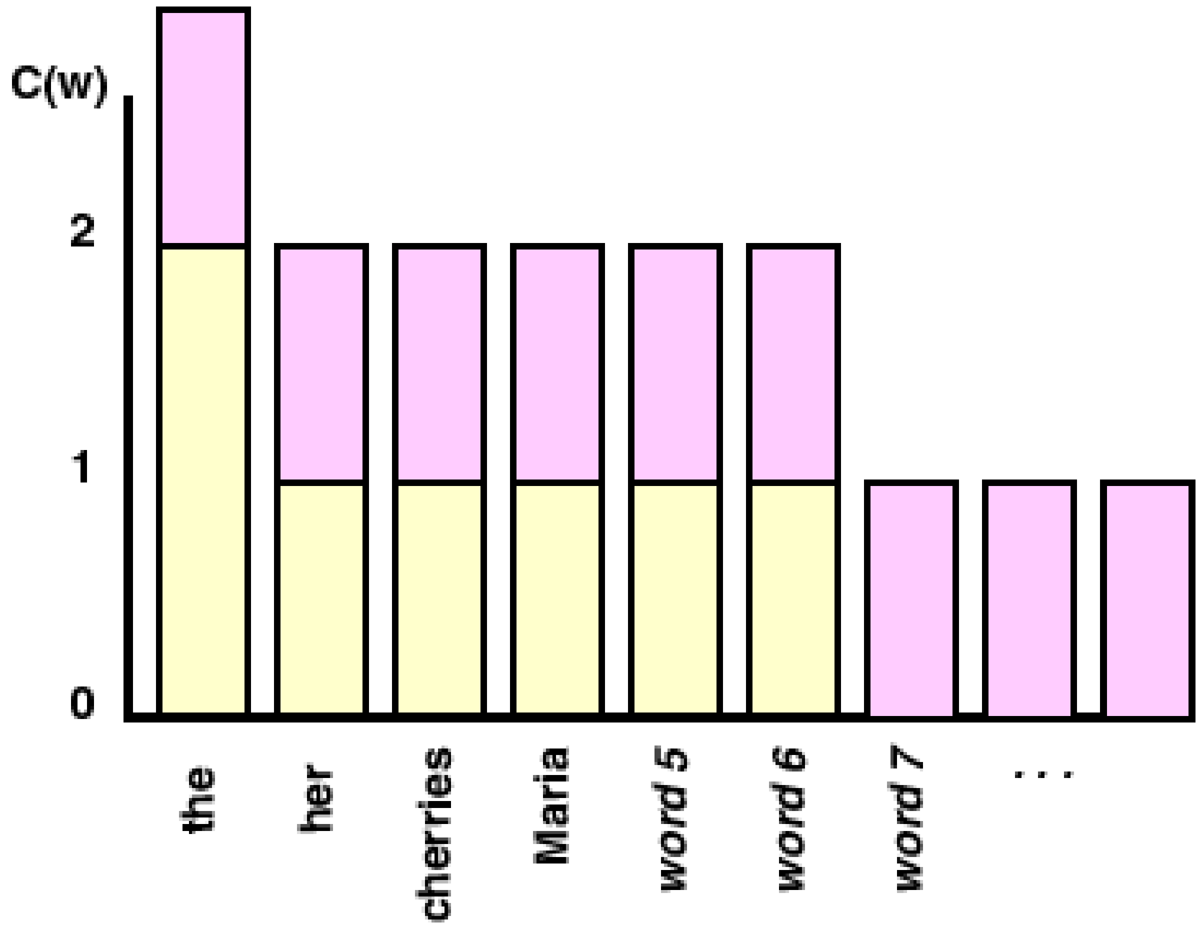
Actual Probability Distribution:



LaPlace’s Law



Laplace’s Law(adding one)



Exercise

MLE: $P_{Lap}(w_1 \dots w_n) = \frac{C(w_1 \dots w_n)}{N}$

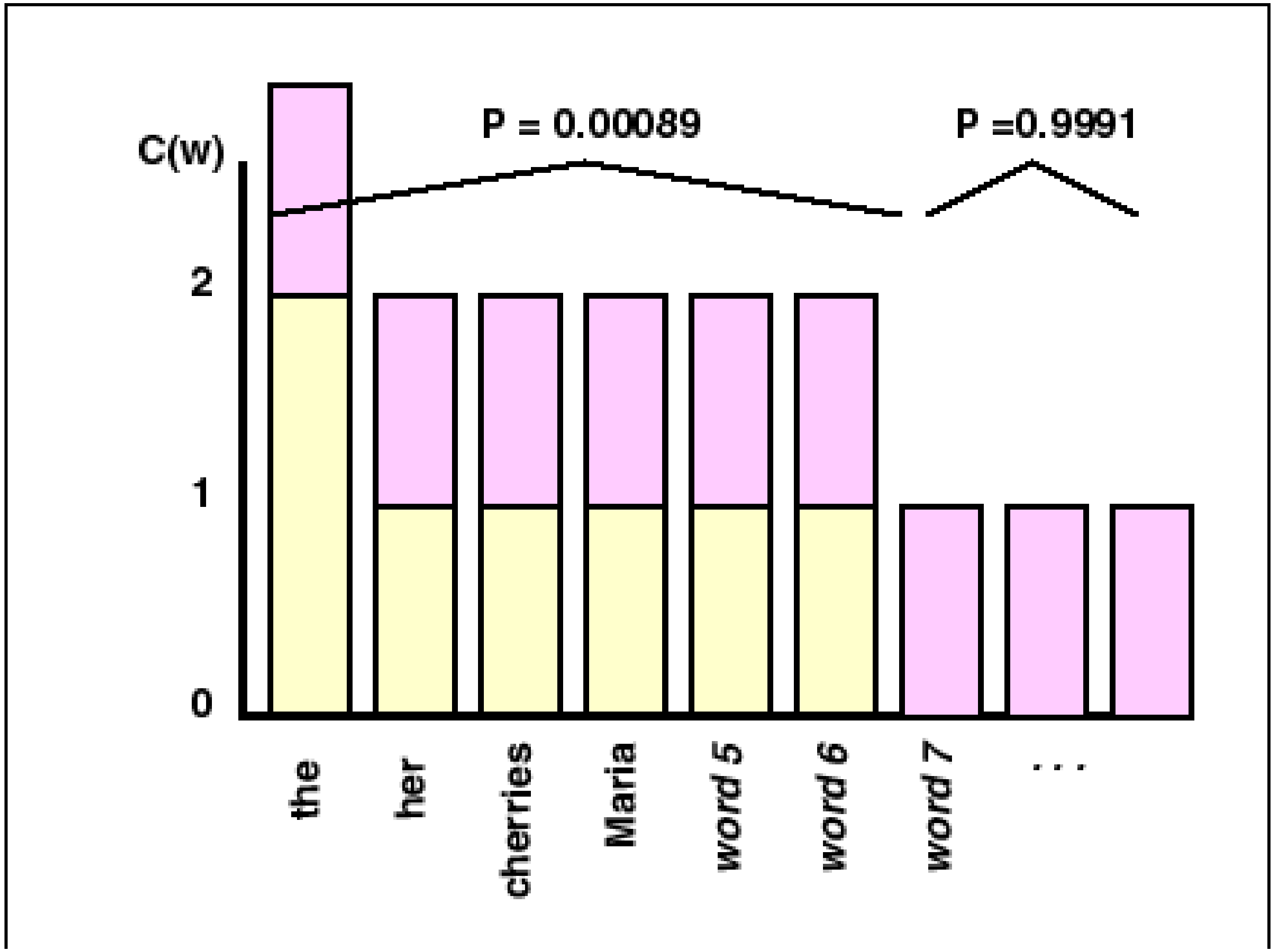
Laplace: $P_{Lap}(w_1 \dots w_n) = ?$

Exercise

MLE: $P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N}$

Laplace: $P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)+1}{N+B}$

Problem of LaPlace’s Law



Lidstone’s Law

- We are adding too much!
- Not add one! But add λ!
- Exercise: $P_{Lid}(w_1 \ldots w_n) = ?$

Lidstone’s Law

- We are adding too much!
- Not add one! But add λ!
- Exercise: $P_{Lid}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)+\lambda}{N+\lambda B}$

summarization

- $P_{Lid}(w_1 \dots w_n) = \frac{C(w_1 \dots w_n) + \lambda}{N + \lambda B}$
  - P = probability of specific n-gram
  - C = count of that n-gram in training data
  - N = total n-grams in training data
  - B = number of “bins” (possible n-grams)
  - λ = small positive number
    - MLE: $\lambda = 0$
      LaPlace’s Law: $\lambda = 1$
      Jeffreys-Perks Law: $\lambda = ½$

Objections to Lidstone's law

- Need an *a priori* way to determine λ.
- Predicts all unseen events to be equally likely
- Gives probability estimates linear in the MLE. Frequency
  - Assume $\mu = \frac{N}{N+B\lambda}$
  - Then $P_{Lid} = \mu \frac{C(w_1 \dots w_n)}{N} + (1-\mu)\frac{1}{B}$

Held out Estimation
Cross Validation

“In Chinese Kungfu Culture, the master always keep the last stance from his student”
– saddnesz.blogspot.com

“In computational linguistics, the master student always keeps a part of the data from the machine learning”
– lilinguistics

In statistical NLP, our data is split into the **training data** and the **held-out data**, the purpose is to compare the probabilities that we get from the 2 different portions to get an estimate.

- You are not assuming a prior weight unlike the λ addition in Laplace and Lidstone smoothening. (Mathematicians! Only trust them if they convince you with the hieroglyphs they are drawing in their formula)
- You are saving your time and effort in getting comparable corpora to compare N-grams (but it's a common fallacy that corpus linguists do, I admit I did the same)
- You are using data from the same source to estimate probabilities of N-grams

Zo, here comes the math:

For each n-gram, $w_1 \cdots w_n$, let:

$$\begin{aligned} C_1(w_1 \cdots w_n) &= \text{frequency of } w_1 \cdots w_n \text{ in training data} \\ C_2(w_1 \cdots w_n) &= \text{frequency of } w_1 \cdots w_n \text{ in held out data} \end{aligned}$$

and recall that N_r is the number of n-grams with frequency r (in the training text). Now let:

$$T_r = \sum_{\{w_1 \cdots w_n : C_1(w_1 \cdots w_n) = r\}} C_2(w_1 \cdots w_n)$$

That is, T_r is the total number of times that all n-grams that appeared r times in the training text appeared in the held out data. Then the average frequency of those n-grams is $\frac{T_r}{N_r}$ and so an estimate for the probability of one of these n-grams is:

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r}{N_r T} \quad \text{where } C_1(w_1 \cdots w_n) = r, \text{ and } T \text{ is the number of}$$

Jibberish?? Obviously this is soooo Greek, sometimes even the people from Greece today don’t understand it.

Imagine we have an ABCorpus that is made up of the letters ‘A’, ‘B’ and ‘C’. So we have $\boldsymbol{vocab_{ABCorpus} = \{A,B,C\}}$ and the ABCorpus looks like this:

|-------------------Training --------------------|-----------Held-Out------|
A B C A B B C C A C B C A A C B C C B C **C B C C B A A C B C A B**

Training Data: T = {A B C A B B C C A C B C A A C B C C B C}

Held Out Data: HO = {C B C C B A A C B C A B}

Fistly, from the **training data**, we list out all the possible 3-grams and group them into how many times they occur (in our data, **T = 11**):

r	**Members of Class$_T$ (r)**	**Count of Class$_T$(N_r)**	**Count of Class$_{HO}$ (T_r)**	**P_{HO}**
3	{CBC}			
2	{BCA, BCC, ACB}			
1	{AAC, ABB, ABC, BBC, CAA, CAB, CAC, CCA, CCB}			
0	{AAA, AAB, ABA, ACA, ACC, BAA, BAB, BAC, BBA, BBB, BCB, CBA, CBB, CCC}			

r	= no. of times the 3-grams appears in **training data**
Class$_T$(r)	= possible 3-grams given vocabulary of the corpus
SizeClass$_T$(Nr)	= no. of **items** the 'set' of Class(r) 3-grams appear in the **training data**
SizeClass$_{HO}$(T_r)	= no. of **times** the 'set' of Class(r) 3-grams appear in the **held-out data**
T	= no. of possible 3-grams in the **held-out data**
P_{HO}	= T_r / (N_r*T)

Fistly, from the **training data**, we list out all the possible 3-grams and group them into how many times they occur (in our data, **T = 11**):

r	**Members of Class_T (r)**	**Count of $\text{Class}_T(N_r)$**	**Count of Class_{HO} (T_r)**	**P_{HO}**
3	{CBC}	1	2	.18
2	{BCA, BCC, ACB}	3	3	.091
1	{AAC, ABB, ABC, BBC, CAA, CAB, CAC, CCA, CCB}			
0	{AAA, AAB, ABA, ACA, ACC, BAA, BAB, BAC, BBA, BBB, BCB, CBA, CBB, CCC}			

r	= no. of times the 3-grams appears in **training data**
$\text{Class}_T(r)$	= possible 3-grams given vocabulary of the corpus
$\text{SizeClass}_T(Nr)$	= no. of **items** the 'set' of Class(r) 3-grams appear in the **training data**
$\text{SizeClass}_{HO}(T_r)$	= no. of **times** the 'set' of Class(r) 3-grams appear in the **held-out data**
T	= no. of possible 3-grams in the **held-out data**
P_{HO}	$= T_r / (N_r * T)$

Fistly, from the **training data**, we list out all the possible 3-grams and group them into how many times they occur (in our data, **T = 11**):

r	**Members of Class$_T$ (r)**	**Count of Class$_T$(N_r)**	**Count of Class$_{HO}$ (T_r)**	**P_{HO}**
3	{CBC}	1	2	.18
2	{BCA, BCC, ACB}	3	3	.091
1	{AAC, ABB, ABC, BBC, CAA, CAB, CAC, CCA, CCB}	9	3	.03
0	{AAA, AAB, ABA, ACA, ACC, BAA, BAB, BAC, BBA, BBB, BCB, CBA, CBB, CCC}	14	3	.019

r	= no. of times the 3-grams appears in **training data**
Class$_T$(r)	= possible 3-grams given vocabulary of the corpus
SizeClass$_T$(Nr)	= no. of **items** the 'set' of Class(r) 3-grams appear in the **training data**
SizeClass$_{HO}$(T_r)	= no. of **times** the 'set' of Class(r) 3-grams appear in the **held-out data**
T	= no. of possible 3-grams in the **held-out data**
P_{HO}	= T_r / (N_r*T)

For each n-gram, $w_1 \cdots w_n$, let:

$$C_1(w_1 \cdots w_n) = \text{frequency of } w_1 \cdots w_n \text{ in training data}$$

$$C_2(w_1 \cdots w_n) = \text{frequency of } w_1 \cdots w_n \text{ in held out data}$$

r	Members of Class$_T$ (r)	Count of Class$_T$(N_r)	Count of Class$_{HO}$ (T_r)	P$_{HO}$
3	{CBC}	1	2	.18
2	{BCA, BCC, ACB}	3	3	.091
1	{AAC, ABB, …, CCB}	9	3	.03
0	{AAA, AAB, …, CCC}	14	3	.019

$$T_r = \sum_{\{w_1 \cdots w_n : C_1(w_1 \cdots w_n) = r\}} C_2(w_1 \cdots w_n)$$

r	Members of Class$_T$ (r)	Count of Class$_T$(N_r)	Count of Class$_{HO}$ (T_r)	P$_{HO}$
3	{CBC}	1	2	.18
2	{BCA, BCC, ACB}	3	3	.09
1	{AAC, ABB, …, CCB}	9	3	.03
0	{AAA, AAB, …, CCC}	14	3	.019

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r}{N_r T} \quad \text{where } C_1(w_1 \cdots w_n) = r\,,$$

and T is the number of tokens.

```
|------------------Training -------------------|---------Held Out-------|
 A B C A B B C C A C B C A A C B C C B C C B C C B A A C B C A B        (3)
```

Fistly, from the **training data**, we list out all the possible 3-grams and group them into how many times they occur (in our data, **T = 11**):

r	**Members of Class$_T$ (r)**	**Count of Class$_T$(N_r)**	**Count of Class$_{HO}$ (T_r)**	**P_{HO}**
3	{CBC}	1	2	.18
2	{BCA, BCC, ACB}	3	3	.091
1	{AAC, ABB, ABC, BBC, CAA, CAB, CAC, CCA, CCB}	9	3	.03
0	{AAA, AAB, ABA, ACA, ACC, BAA, BAB, BAC, BBA, BBB, BCB, CBA, CBB, CCC}	14	3	.019

Why doesn't this sum up to 1????

Fistly, from the **training data**, we list out all the possible 3-grams and group them into how many times they occur (in our data, **T = 11**):

r	Members of Class_T (r)	Count of $\text{Class}_T(N_r)$	Count of Class_{HO} (T_r)	P_{HO}	
3	{CBC}	1	2	.18	.18*1 = .18
2	{BCA, BCC, ACB}	3	3	.091	.09*3 = .27
1	{AAC, ABB, …, CCB}	9	3	.030	.03*9 = .27
0	{AAA, AAB, ABA, ACA, ACC, BAA, …, CCC}	14	3	.019	.019*14=.272727272727

Because this adds to 1!!!!

Class_T(r), using the **training data** we are listing the different *variations of the 3-grams* *and* ranking them by *how often they occurred in current text*

N_r , we count the *no. of times 3-grams appears in* ***'seen'*** *text* according to the ranks

T_r , we count the *no. of times 3-grams appears in* ***'unseen'*** *text* according to the ranks

P_{HO} , we calculate the ***estimated*** *probability of 3-grams will appear* ***in general*** according to their ranks

This is the REAL **Phở** >>>>>>>>>>>>>>>>

- **Test data**, a ***CARDINAL SIN*** in Stats NLP is to test on your training data.
  - Honestly, if you think that that will give you 100% precision, it won't. More often there are bugs in our systems that even if you train and test on the same data it doesn't give you 100%. (From experience)

- **Overtraining**, models (i.e. in our case N-grams like events) tends to expect future events to be like the seen events
  - Language in general shouldn’t work that way, in Singlish there are phrases like 'Act Blur' (POS tagging) 'act as....', 'act like he is ...', 'act as ...’ However statistically, it does if you make sure that your training data does cover an optimal amoun of patterns/parameters.

- So **predictive natured Stats NLP** may be necessaire, ML methods are in Part III of Manning's book. Other stats methods are in the rest of this chapter
  - Personal fav. ML is HMM, because it has ICE CREAM and it’s simple to implement, but maths and computer people like to make complexed-up HMM to torture NLP students

- Training, Held-out, Test data are **independent**. So with the training data you train the model, then with the held-out data you estimate the effectiveness (F-score) of your model, then ...

```
While (F-score < satisfactory) //s…factory sounds like socola
{
  TrainModel(TrainingData) //returns updated Model, not
                           //the MeganFox kind
  EstimateF-score(Model, Held-outData)
  If F-score > satisfactory{break;}
 }
EstimateF-score(Model, TestData) // returns updated F-score
WritePaperandSubmit(F-score, Model, buckloads of pride)
```

- Then, the chapter goes on and on about how you *test your systems using t-test* so that you can know which systems are significantly better…
  - yada, yada… we've heard enough on t-test, significant testing, confidence interval…yada, yada

- If we can perform **hold-out estimates** between *hold-on and training data*, how about between *hold-out and test data*?

Cross-validation = Deleted estimation

(I don’t really care for fancy synonyms)

Jelinek and Mercer (1985) use a form of two-way cross-validation that they call *deleted estimation*. Suppose we let N_r^a be the number of n-grams occurring r times in the a^{th} part of the training data, and T_r^{ab} be the total occurrences of those bigrams from part a in the b^{th} part. Now depending on which part is viewed as the basic training data, standard held out estimates would be either:

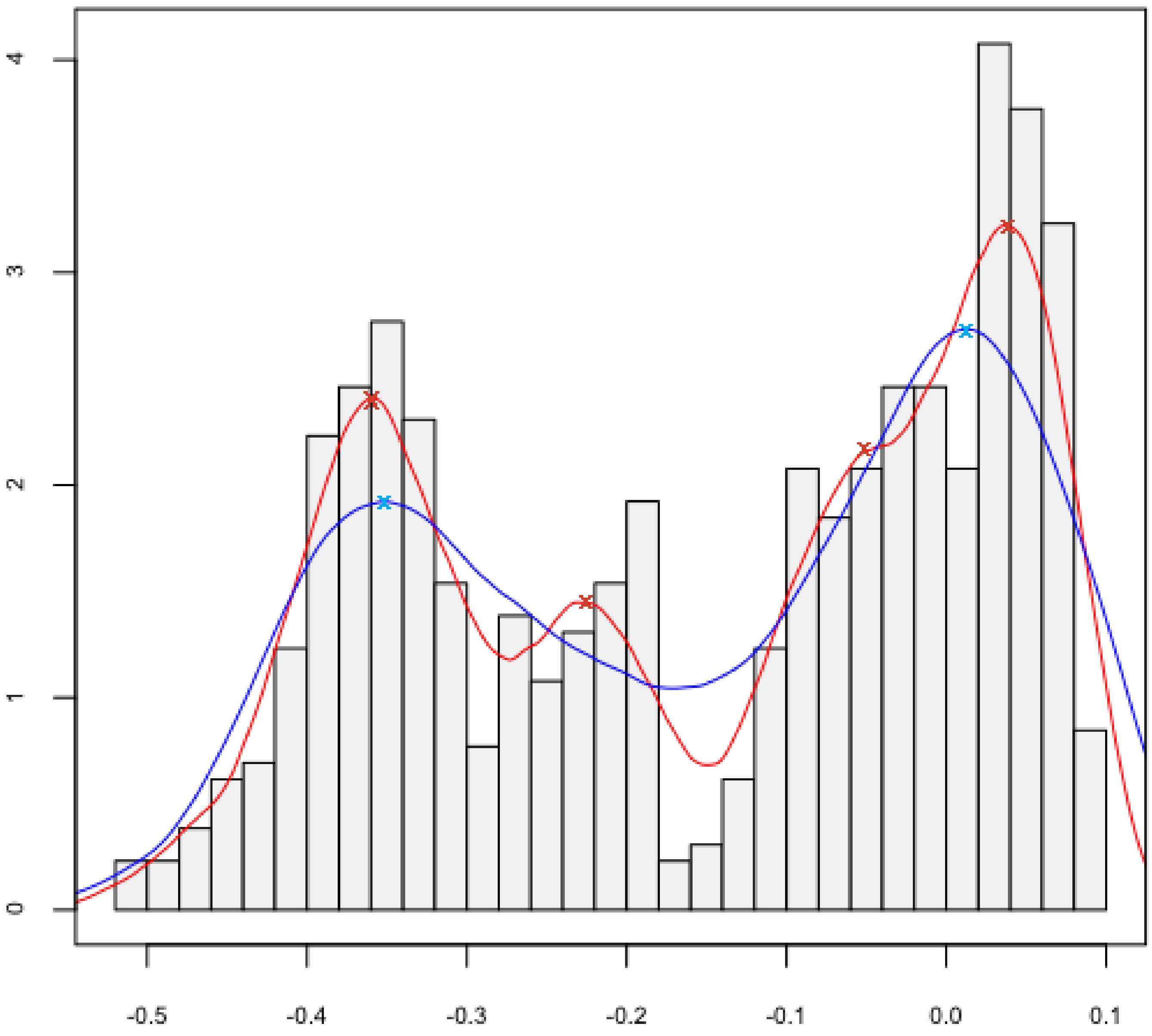
$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r^{01}}{N_r^0 N} \text{ or } \frac{T_r^{10}}{N_r^1 N} \qquad \text{where } C(w_1 \cdots w_n) = r$$

The more efficient deleted interpolation estimate does counts and smoothing on both halves and then does a weighted average of the two:

$$P_{\text{del}}(w_1 \cdots w_n) = \frac{T_r^{01} + T_r^{10}}{N(N_r^0 + N_r^1)} \qquad \text{where } C(w_1 \cdots w_n) = r$$

Still, this looks very Greeky…

Probability Curve Smoothening **!=** facials at spas



Cross-validation = Deleted estimation

(Once again, I still don't really care for fancy synonyms)

|--------Train1---------|---------Train2 -------|-----------Held-Out---------|

A B C A B B C C A C B C A A C B C C B C **C B C C B A A C B C A B C**

$$P_{\text{ho}}(w_1 \cdots w_n) = \frac{T_r^{01}}{N_r^0 N} \text{ or } \frac{T_r^{10}}{N_r^1 N} \qquad \text{where } C(w_1 \cdots w_n) = r$$

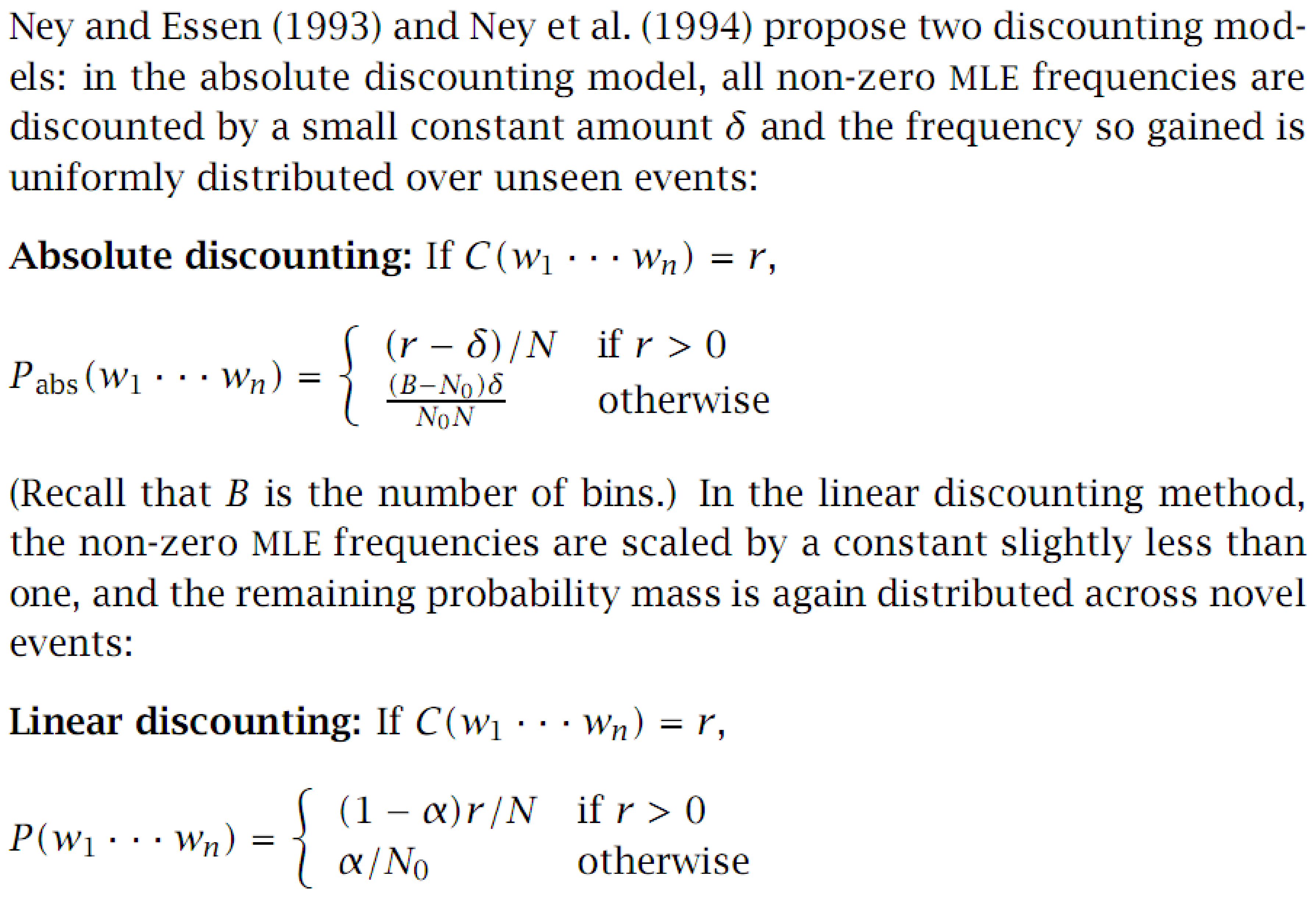
$$\frac{Pho1}{sizeTrain1 * TrainAll} + \frac{Pho2}{sizeTrain2 * TrainAll}$$

$$= \frac{Pho1 + Pho2}{TrainAll(sizeTrain1 + sizeTrain2)}$$

$$\downarrow$$

$$P_{\text{del}}(w_1 \cdots w_n) = \frac{T_r^{01} + T_r^{10}}{N(N_r^0 + N_r^1)} \qquad \text{where } C(w_1 \cdots w_n) = r$$

Discounting != Bargainning at Aldi or Audi or A*di (eeewww, regex)



Ney and Essen (1993) and Ney et al. (1994) propose two discounting models: in the absolute discounting model, all non-zero MLE frequencies are discounted by a small constant amount δ and the frequency so gained is uniformly distributed over unseen events:

Absolute discounting: If $C(w_1 \cdots w_n) = r$,

$$P_{\text{abs}}(w_1 \cdots w_n) = \begin{cases} (r-\delta)/N & \text{if } r > 0 \\ \frac{(B-N_0)\delta}{N_0 N} & \text{otherwise} \end{cases}$$

(Recall that B is the number of bins.) In the linear discounting method, the non-zero MLE frequencies are scaled by a constant slightly less than one, and the remaining probability mass is again distributed across novel events:

Linear discounting: If $C(w_1 \cdots w_n) = r$,

$$P(w_1 \cdots w_n) = \begin{cases} (1-\alpha)r/N & \text{if } r > 0 \\ \alpha/N_0 & \text{otherwise} \end{cases}$$

Yet another Greeky looking section…

Absolute Discounting

(Once again, we use arrowy Greek to translate Greek into understandable thingy)

Good-Turing Estimator: If $C(w_1 \cdots w_n) = r > 0$,

$$P_{\mathrm{GT}}(w_1 \cdots w_n) = \frac{r^*}{N} \quad \text{where } r^* = \frac{(r+1)S(r+1)}{S(r)}$$

$$P_{\mathrm{abs}}(w_1 \cdots w_n) = \begin{cases} (r-\delta)/N & \text{if } r > 0 \\ \frac{(B-N_0)\delta}{N_0 N} & \text{otherwise} \end{cases} \qquad \delta \approx 0.77$$

If $C(w_1 \cdots w_n) = 0$,

$$P_{\mathrm{GT}}(w_1 \cdots w_n) = \frac{1 - \sum_{r=1}^{\infty} N_r \frac{r^*}{N}}{N_0} \approx \frac{N_1}{N_0 N}$$

Linear Discounting

(Once again, we use arrowy Greek to translate Greek into understandable thingy)

Good-Turing Estimator: If $C(w_1 \cdots w_n) = r > 0$,

$$P_{\mathrm{GT}}(w_1 \cdots w_n) = \frac{r^*}{N} \quad \text{where } r^* = \frac{(r+1)S(r+1)}{S(r)}$$

$$P(w_1 \cdots w_n) = \begin{cases} (1-\alpha)r/N & \text{if } r > 0 \\ \alpha/N_0 & \text{otherwise} \end{cases}$$

If $C(w_1 \cdots w_n) = 0$,

$$P_{\mathrm{GT}}(w_1 \cdots w_n) = \frac{1 - \sum_{r=1}^{\infty} N_r \frac{r^*}{N}}{N_0} \approx \frac{N_1}{N_0 N}$$

More complex Discounting

(Yet Another Discounting of Estimation - YADE)

Ristad (1995) explores the hypothesis that natu-
ral sequences use only a subset of the possible bins. He derives various forms for a *Natural Law of Succession*, including the following probability estimate for an n-gram with observed frequency $C(w_1 \cdots w_n) = r$:

$$P_{\text{NLS}}(w_1 \cdots w_n) = \begin{cases} \frac{r+1}{N+B} & \text{if } N_0 = 0 \\ \frac{(r+1)(N+1+N_0-B)}{N^2+N+2(B-N_0)} & \text{if } N_0 > 0 \text{ and } r > 0 \\ \frac{(B-N_0)(B-N_0+1)}{N_0(N^2+N+2(B-N_0))} & \text{otherwise} \end{cases}$$

Good-Turing Estimation

Good-Turing Discounting

- What's this?
- To use the count of things we've seen once to help estimate the count of things we've never seen
- Formula for Good-Turing algorithm

Formalizing The Good-Turing Algorithm

- Smoothed Count C*:

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

- To estimate total number of unseen types:

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

GT Fish Example

- **Imagine you are fishing**

There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass

- **You have caught 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish**
- **How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?**

  3/18

- **Assuming so, how likely is it that next species is trout?**

  Must be less than 1/18

Good-Turing

- **Notation: Nx is the frequency-of-frequency-x**

So N10=1

Number of fish species seen 10 times is 1 (carp)

- N1=3

Number of fish species seen 1 is 3 (trout, salmon, eel)

- **To estimate total number of unseen species** Use number of species (words) we've seen once

$c_0^* = c_1$ $p_0 = N_1/N$ $c^* = (c+1)\frac{N_{c+1}}{N_c}$

- **All other estimates are adjusted (down) to give for unseen Slide from**

Good-Turing

- **Notation: Nx is the frequency-of-frequency-x**

  So $N_{10}=1$, $N_1=3$, etc

- **To estimate total number of unseen species**

  Use number of species (words) we've seen once

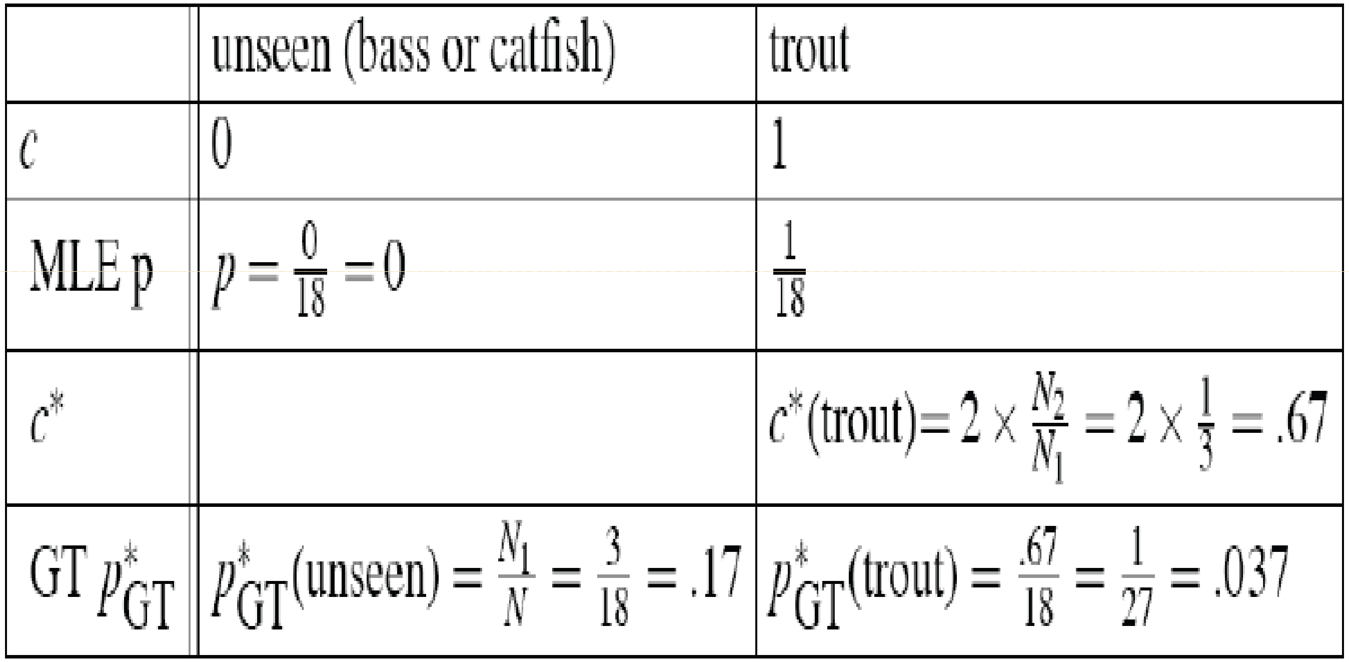
  $c_0^* = c_1$ $p_0 = N_1/N$ $p_0=N_1/N=3/18$

  $$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

- **All other estimates are adjusted (down) to give probabilities for unseen**

  $$c^* = (c+1)\frac{N_{c+1}}{N_c}$$ $P(eel) = c^*(1) = (1+1)\ 1/3 = 2/3$

GT Fish Example



	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Complications

- In practice, assume large counts (c>k for some k) are reliable:

$$c^* = c \quad \text{for } c > k$$

- That complicates c*, making it:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \quad \text{for } 1 \leq c \leq k.$$

Complications

- **Also: we assume singleton counts c=1 are unreliable, so treat Ngrams with count of as if they were count=0**
- **Also, need the Nk to be non-zero, so we need to smooth (interpolate) the Nk counts before computing c* from them**