Language Modeling

Introduction to N-grams

Probabilistic Language Models

- Today's goal: assign a probability to a sentence

Why?

- Machine Translation: 希望輸出的機率最高者為正確答案
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!!

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$ 要知道這整句話出現的機率
  就要計算每一個字出現的條件機率

- Related task: probability of an upcoming word:

  $P(w_5 | w_1,w_2,w_3,w_4)$ 當前四個字確定時
  第五個字為w5的機率

- A model that computes either of these:

  $P(W)$ or $P(w_n | w_1,w_2 \ldots w_{n-1})$ 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(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

  $P(A|B) = P(A,B) / P(B)$ Rewriting: $P(A|B)P(B) = P(A,B)$

- 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

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

簡單來講
每一個字出現的條件機率的Given條件，就是比這個字早出現的所有字的機率乘積

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

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

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

- Could we just count and divide?

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

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

不可能計算某個句子的總出現次數
因為這些字要以相同順序出現在同一句的可能性太低了
也就是說資料量會不足

解決方法：Markov Assumption

Dan Jurafsky

Markov Assumption

Andrei Markov

- Simplifying assumption:

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

而不是P(the|that,transparent,so,is,water,its)

- Or maybe

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

一個字出現的條件機率，他的Given條件不需要先前所有字的機率乘積
而是可以選最後一個字或最後兩個字
（假設這個字只和他的前1~2相關）

Markov Assumption

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

- In other words, we approximate each component in the product

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

只計算last few terms

Dan Jurafsky

Simplest case: Unigram model

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

Some automatically generated sentences from a unigram model

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

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

that, or, limited, the
```

假設每個字出現的機率是獨立的
也就是第i個字的出現和第i-1, i-2 ... i-k無關
所以產生出來的句子很糟，不像是一個完整句子

Dan Jurafsky

Bigram model

- Condition on the previous word:

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

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

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

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

假設每個字出現的機率是只和他的前一個字有關
也就是第i個字的出現只和第i-1個字有關
所以產生出來的句子很糟，不像是一個完整句子
最多會有幾個看起來合理的複合名詞
e.g. new car, car parking …

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.” 主詞接的真正動詞可能距離很遠，例如這邊的 computer crashed

- But we can often get away with N-gram models

Language Modeling

Introduction to N-grams

Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

- The Maximum Likelihood Estimate

當我們看到w(i–1)時，有多少比例他的下一個字會接w(i)?

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

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

An example

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

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

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	$\boxed{827}$	0	9	0	0	0	2
want	2	0	$\boxed{608}$	1	6	6	5	1
to	2	0	4	$\boxed{686}$	2	0	6	211
eat	0	0	2	0	16	2	$\boxed{42}$	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: 簡單講就是總出現次數

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	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

827/2533 = 0.33

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

What kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

代表want Chinese出現次數比want English頻繁

代表很常出現...want to...

代表沒出現過 ...spend want ...

Practical Issues

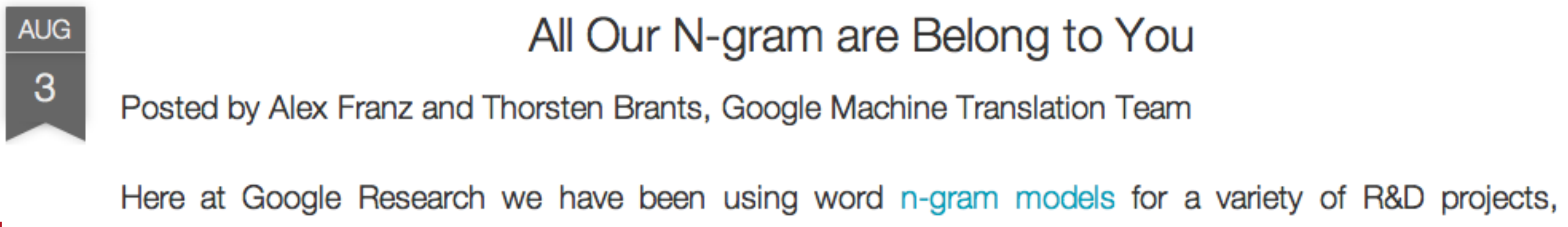
- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Language Modeling Toolkits

- SRILM
  - http://www.speech.sri.com/projects/srilm/

Google N-Gram Release, August 2006



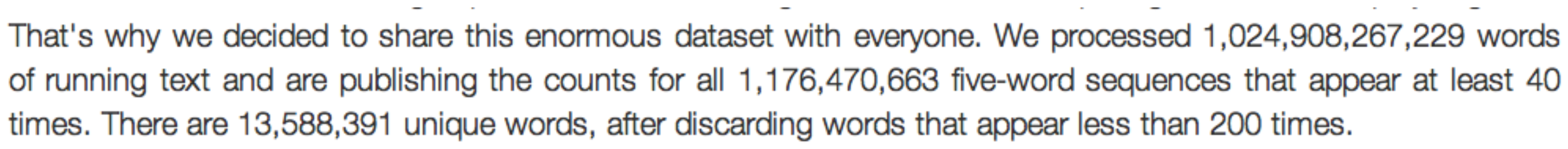
AUG 3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

…



That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

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

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

Google Book N-grams

- http://ngrams.googlelabs.com/

Language Modeling

Estimating N-gram Probabilities

Language Modeling

Evaluation and Perplexity

Evaluation: How good is our model?

好的結果的出現機率 > 不好的結果的出線機率

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- 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 that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

Intuition of Perplexity

像第一句和第三句，根本沒有標準答案，就很難預測

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

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)

mushrooms	0.1
pepperoni	0.1
anchovies	0.01
….	
fried rice	0.0001
….	
and	1e-100

- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

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

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

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

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

The Shannon Game intuition for perplexity

- From Josh Goodman
- How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
  - Perplexity 10
- How hard is recognizing (30,000) names at Microsoft.
  - Perplexity = 30,000
- If a system has to recognize
  - Operator (1 in 4)
  - Sales (1 in 4)
  - Technical Support (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 53
- Perplexity is weighted equivalent branching factor

Perplexity as branching factor

- Let’s suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned}\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= (\frac{1}{10}^{N})^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10\end{aligned}$$

假設有一個字串
長度=N，只包含0-9
每一個數字出現在機率都是1/10
那就可以用左邊的算式算出：

```
  P(w1,w2,…,wN)^(1/N)
= P(1/10 * 1/10 * … * 1/10) ^(1/N)
= (1/10^N) ^ (1/N)
= (1/10)^(-1)
= 10
```

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity越低
預測的結果會越準確

Language Modeling

Evaluation and Perplexity

Language Modeling

Generalization and zeros

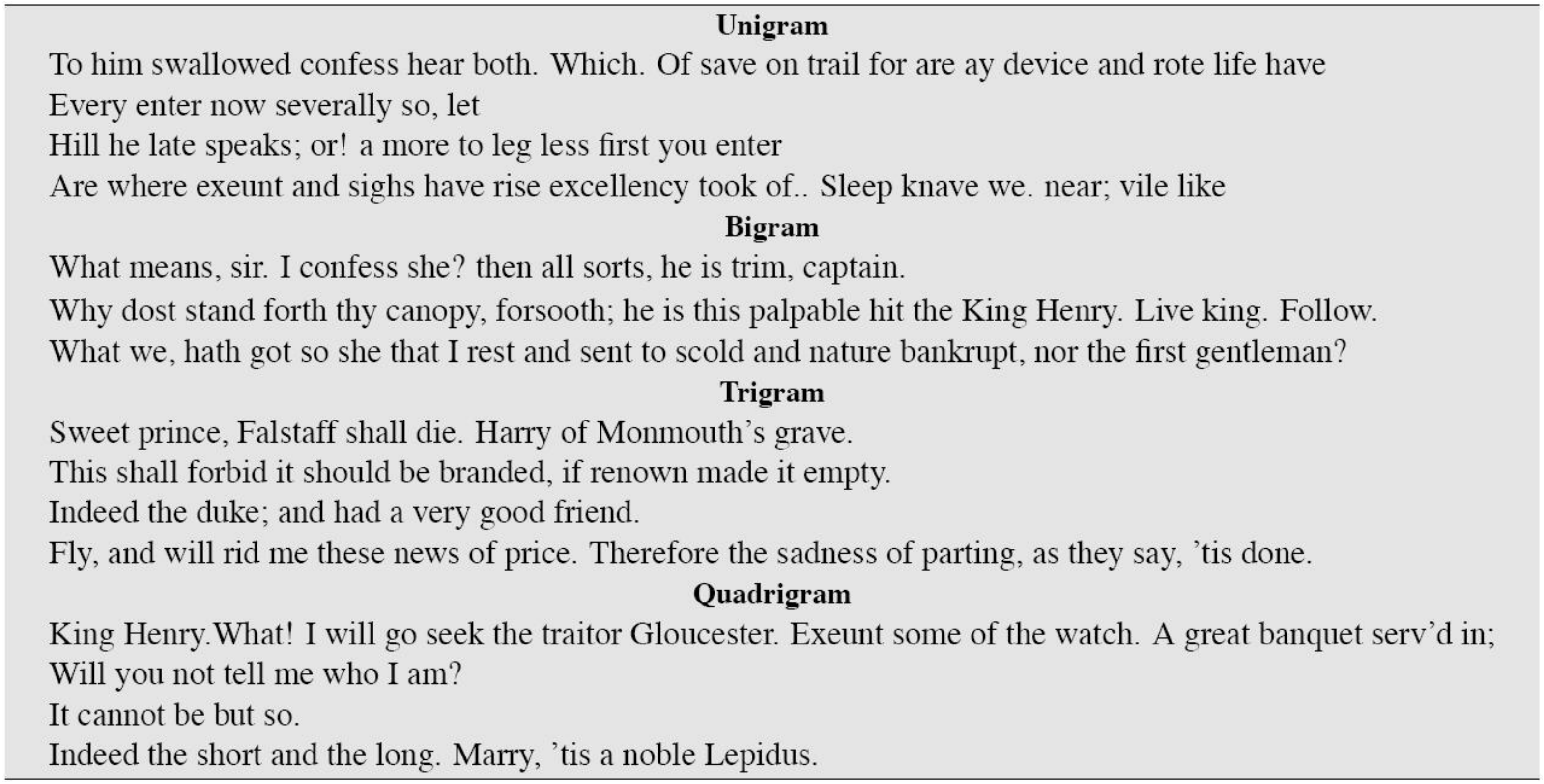
The Shannon Visualization Method

- 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
```

簡單講就是從第一個字開始
透過Bigram
一直選出下一個可能的字
最後組成一個機率最高的句子

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

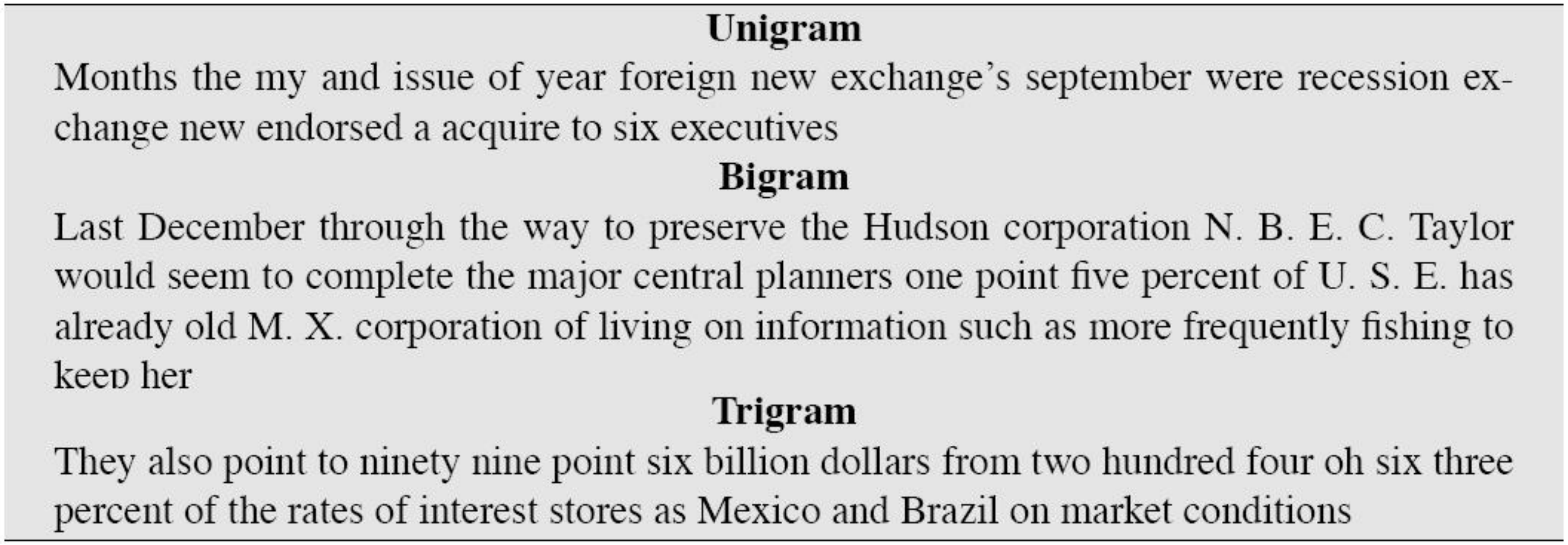
King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

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

Quadrigrams缺點：
這些長句子可能只在Shakespeare出現
如果用其他句子當testing data可能效果就不會很好

The wall street journal is not shakespeare (no offense)



Unigram

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

Bigram

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

Trigram

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

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

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)!

Training data沒出現過的bigram
他們的機率就會被設為0
但沒出現過並不代表該文法是錯的
在資料不夠的情況下，很可能會把正確的testing data誤判為0

解法：Add one smoothing
把所有bigram的出現次數都+1
避免0的發生

Language Modeling

Generalization and zeros

Language Modeling

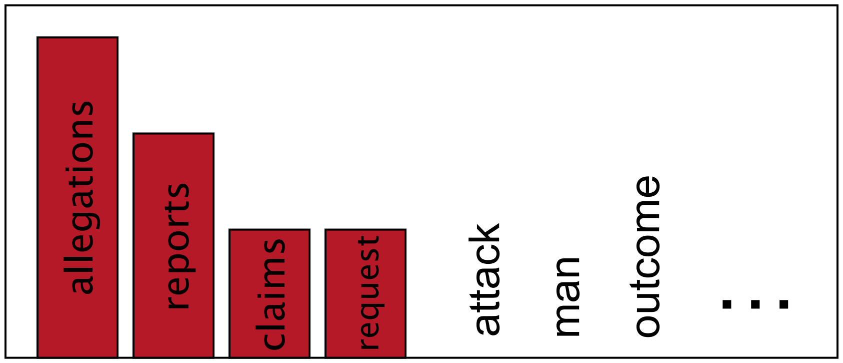
Smoothing: Add-one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request

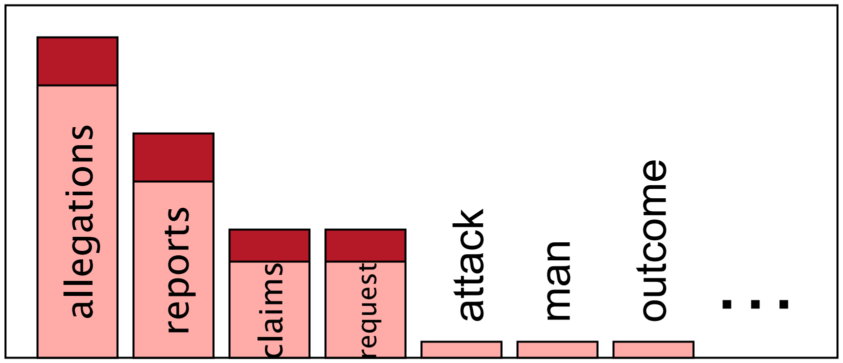
  7 total



- Steal probability mass to generalize better

  P(w | denied the)
  2.5 allegations
  1.5 reports
  0.5 claims
  0.5 request
  2 other

  7 total



Dan Jurafsky

Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$
- Add-1 estimate: $$\boxed{P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}}$$

分母為何+V?
總共有V種bigram

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.

bagel在training data的出現機率是0.0004
但並不代表bagel在testing data的出現機率也會是0.0004
誰知道testing data長怎樣？
不過我們還是得假設他的出現機率是0.0004
因為沒有更好的選擇了

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

Dan Jurafsky

參考Bigram Example

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	$\boxed{0.21}$	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

0.21 = 828/4000

0.00025 = 1/4000

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

Dan Jurafsky

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

Language Modeling

Smoothing: Add-one (Laplace) smoothing

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Dan Jurafsky

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram
- Interpolation works better

已經有了trigram
想要透過現有trigram
轉換為bigram, unigram

Interpolation > Backoff

Linear Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

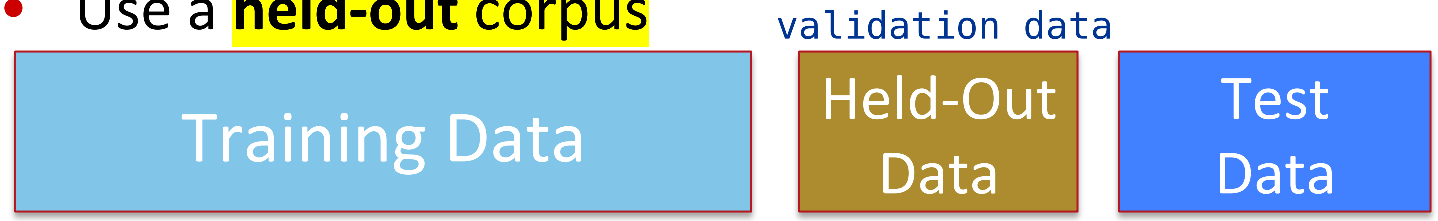
- Lambdas conditional on context:

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

Dan Jurafsky

How to set the lambdas?

- Use a **held-out** corpus



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

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

拿training data去訓練n-gram
找到一個λ，使得所有n-gram的mix能夠在held-out data得到最高的機率

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

如果在訓練n-gram時遇到沒出現過的字
就把它轉為<UNK>這個token

Huge web-scale n-grams

加快速度的方法

- 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 移除那些只有出現一次的字
  - Entropy-based pruning
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for Web-scale N-grams

Add-one smoothing不適用於Language Model

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

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

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

當我們看到w(i−1) ~ w(i−k+1)時，下一個字是w(i)的機率有多少？
(k可以自訂，代表的意義是k−gram，代表要看k個前綴字)
1. 如果這筆資料存在，即count(w(i, i−k+1))>0，那就直接用likelihood去算
2. 如果這筆資料不在，即count(w(i, i−k+1))=0，那就退到k−1 gram，並乘上某個常數去計算

N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid backoff

Advanced Language Modeling

- Discriminative models:
  - choose n-gram weights to improve a task, not to fit the training set
- Parsing-based models
- Caching Models
  - Recently used words are more likely to appear 最近用過的字會比較容易再出現

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2}w_{i-1}) + (1-\lambda)\frac{c(w \in history)}{\mid history \mid}$$

  - These perform very poorly for speech recognition (why?)

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Language Modeling

Advanced: Good Turing Smoothing

Reminder: Add-1 (Laplace) Smoothing

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

More general formulations: Add-k

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

m = kv

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

Unigram prior smoothing

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

unigram probability

$$P_{\text{UnigramPrior}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell
- Use the count of things we've **seen once**
  - to help estimate the count of things we've **never seen**

用出現過的資料
去推測沒有出現過的資料

Notation: N_c = Frequency of frequency c

- N_c = the count of things we've seen c times
- Sam I am I am Sam I do not eat

I	3
sam	2
am	2
do	1
not	1
eat	1

Ni = k -> 出現i次的字有k個

$N_1 = 3$

$N_2 = 2$

$N_3 = 1$

Dan Jurafsky

Good-Turing smoothing intuition

- You are fishing (a scenario from Josh Goodman), and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that next species is new (i.e. catfish or bass)
  - Let's use our estimate of things-we-saw-once to estimate the new things.
  - 3/18 (because N_1=3) 因為下一個new species一定只出現一次
    所以就把N1=3當作是new species的出現機率
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18
  - How to estimate?

如果用和next species is new相同的算法去計算next species is trout的機率
那就不再會是1/18了

Good Turing calculations

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

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

- Unseen (bass or catfish)
  - c = 0:
  - MLE p = 0/18 = 0 Maximum likelihood
  - P^*_{GT} (unseen) = N_1/N = 3/18

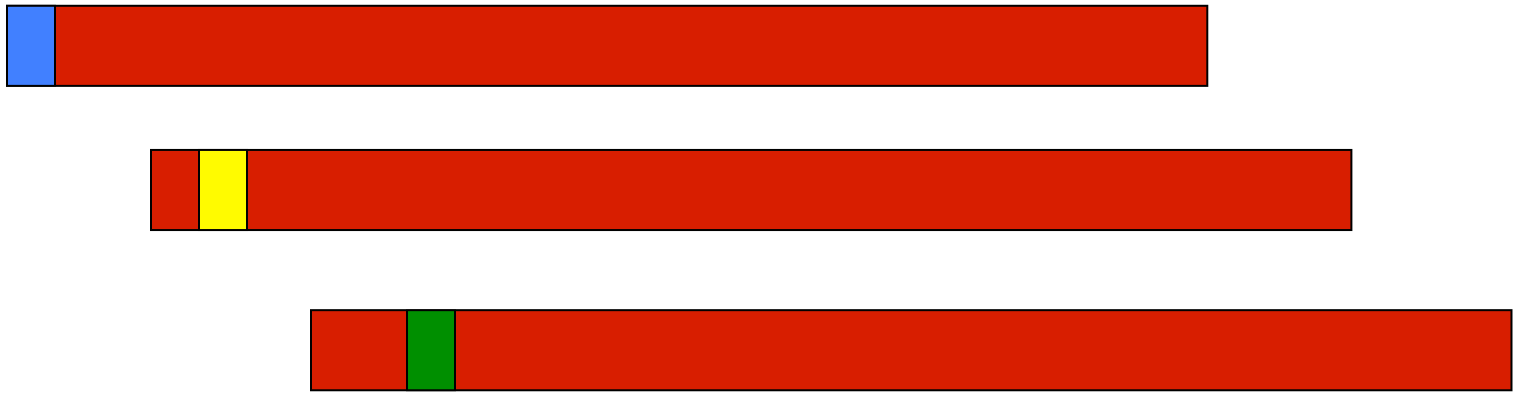
next species is new
出現的機率為N1/N
因為next species is new一定只會有一個

- Seen once (trout)
  - c = 1
  - MLE p = 1/18
  - C*(trout) = 2 * N2/N1
    = 2 * 1/3
    = 2/3
- P^*_{GT}(trout) = 2/3 / 18 = 1/27

因為如果再出現一次trout
就代表所有魚裡面會有兩個trout
所以要把N2的機率也拿去算

Ney et al.'s Good Turing Intuition

H. Ney, U. Essen, and R. Kneser, 1995. On the estimation of 'small' probabilities by leaving-one-out. IEEE Trans. PAMI. 17:12,1202-1212



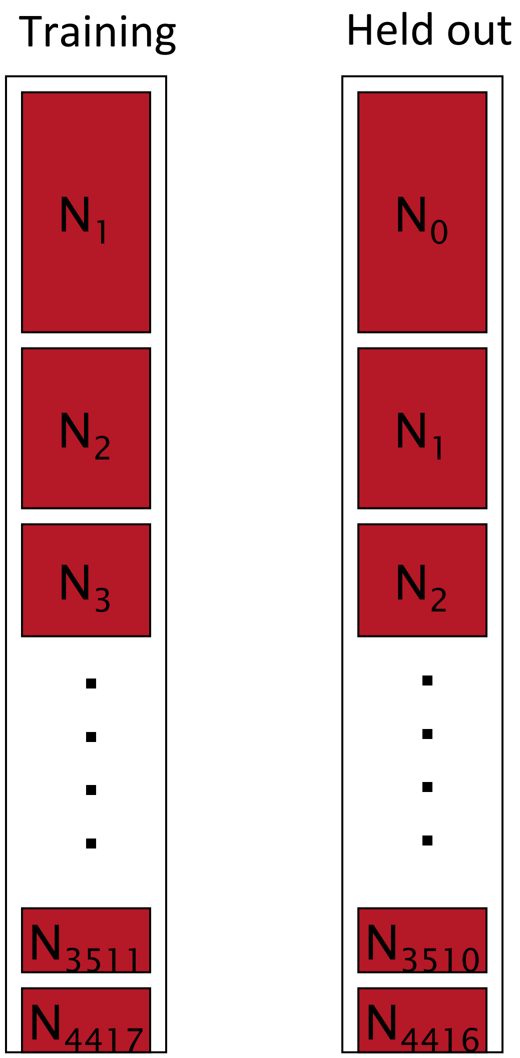
Held-out words:

假設整句的總長度為C
對於整句話的每一個單字
我們會拿其他C-1個單字做N-gram去預測該單字
跑完C次迴圈之後
一樣可以得到長度為C句子
但這個句子的內文是透過各自單字的其他C-1個單字預測出來的

Ney *et al.* Good Turing Intuition (slide from Dan Klein)

- Intuition from leave-one-out validation
  - Take each of the *c* training words out in turn
  - *c* training sets of size *c*–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen *k* times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count *k*
  - There are N_k words with training count *k*
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - …or expected count:

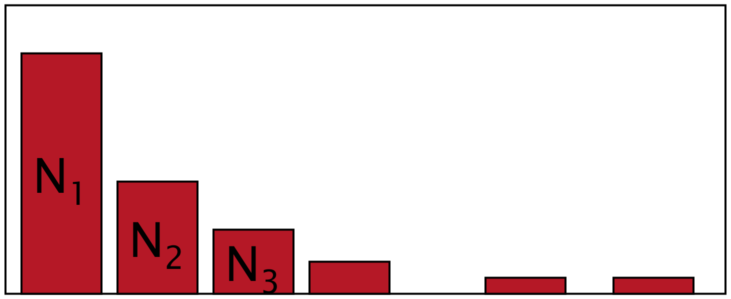
$$k^* = \frac{(k+1)N_{k+1}}{N_k}$$



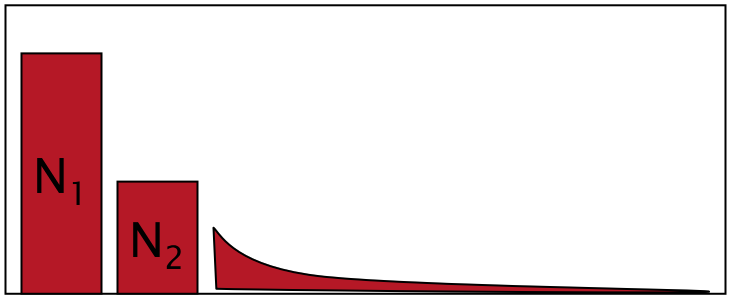
如果要計算一個已經出現過K次的字的出現機率
那就要把N(k)和N(k+1)一起拿去計算

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



如果要計算出現4417次的the的出現機率
那就要拿N4417和N4418丟進去計算
但因為資料集中並沒有N4418的資料
所以沒辦法直接這樣算
解決方法：
把離散的統計資料透過某個best-fit power law
轉換成連續的資料
這樣就不用擔心某一個k是沒有資料的

Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

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

經過best-fit的結果

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Language Modeling

Advanced: Good Turing Smoothing

Language Modeling

Advanced: Kneser-Ney Smoothing

Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

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

- It sure looks like c* = (c - .75)

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram P(w)?

面對測試資料沒有出現在訓練資料的預設方法

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading*___Francisco / glasses___?
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"

因為Franciso比glasses常出現
所以如果用傳統unigram
會得到Franciso

- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{continuation}(w)$: "How likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

這個公式可以想成
w(i-1), w這樣的bigram組合總共出現了幾種？
e.g. w=food
Chinese food, English food, English food,
Japanese food

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing III

- Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- normalized by the # of words preceding all words:

$$\boxed{P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}}$$

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

避免大量出現的San-Franciso
高估了Franciso的bigram分數

Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Language Modeling

Advanced: Kneser-Ney Smoothing