04 N-GRAM LANGUAGE MODELS

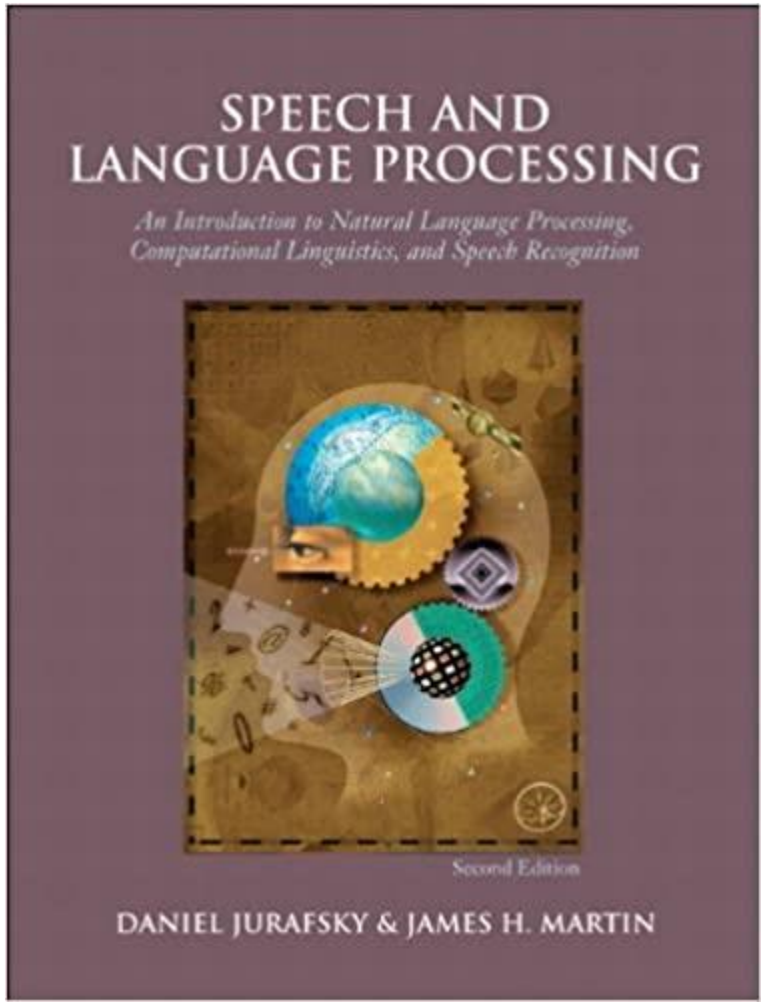
Spring 2023

CS6431 Natural Language Processing

B1:

Speech and Language Processing (Third Edition draft – Jan2022)

Daniel Jurafsky, James H. Martin



Credits

1. B1
2. https://machinelearningmastery.com/what-is-maximum-likelihood-estimation-in-machine-learning/

Assignment

Read:

B1: Chapter 3

Problems: Exercise problems of Chapter 3

Outline

- N-gram language model
- Evaluating language models
- Sampling sentences from a language model

N-gram language model

Next Word Prediction

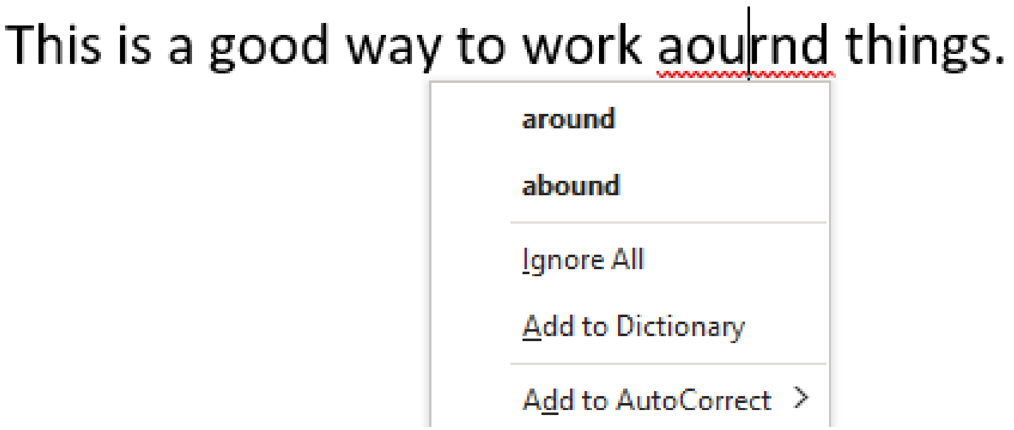
- Please turn your homework ...
  - Likely words: ‘in’, ‘over’
  - Not ‘refrigerator’, ‘the’
- Assigning probability to each possible next word
- Assigning probability to an entire sentence
  - ‘all of a sudden I notice three guys standing on the sidewalk’

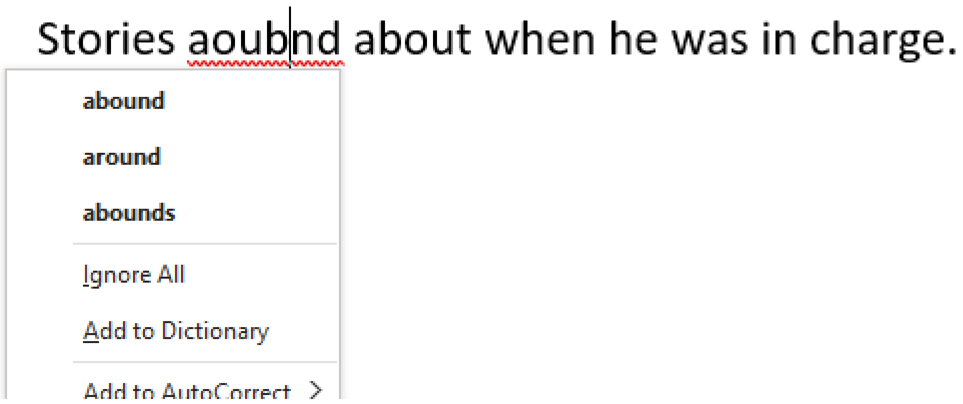
  Vs.

  - ‘on guys all I of notice sidewalk three a sudden standing the’

Why assign probabilities?

- To identify words in noisy, ambiguous input
  - Speech Recognition
    - ‘got it!’ vs ‘gutted!’
  - Spelling correction or grammatical error correction





Machine Translation

他	向	记者	介绍了	主要	内容
He	to	reporters	introduced	main	content

Set of potential translations

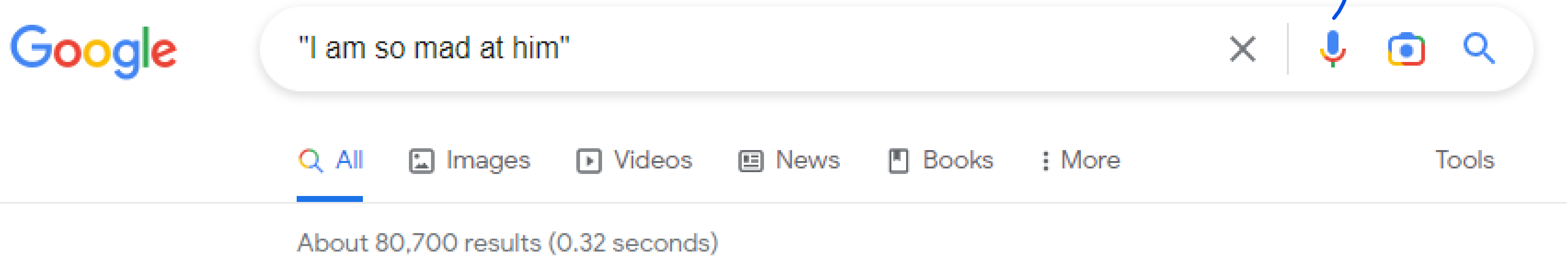
he introduced reporters to the main contents of the statement
he briefed to reporters the main contents of the statement
he briefed reporters on the main contents of the statement

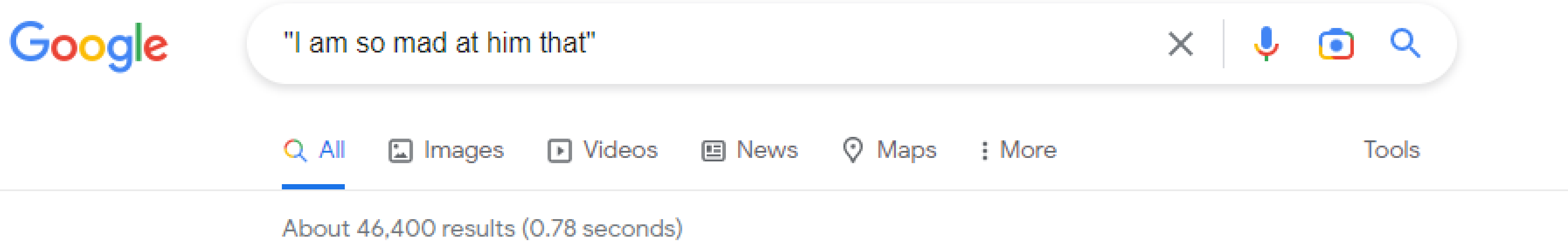
Language Models (LMs)

- LMs: Models that assign probabilities to sequences of words
- ***n-gram*** (model): a sequence of *n* words.
  - Bigram just previous word
  - Trigram
- n-gram models estimate the probability of the last word of an n-gram given the previous words
  - Also to assign probabilities to entire sequences

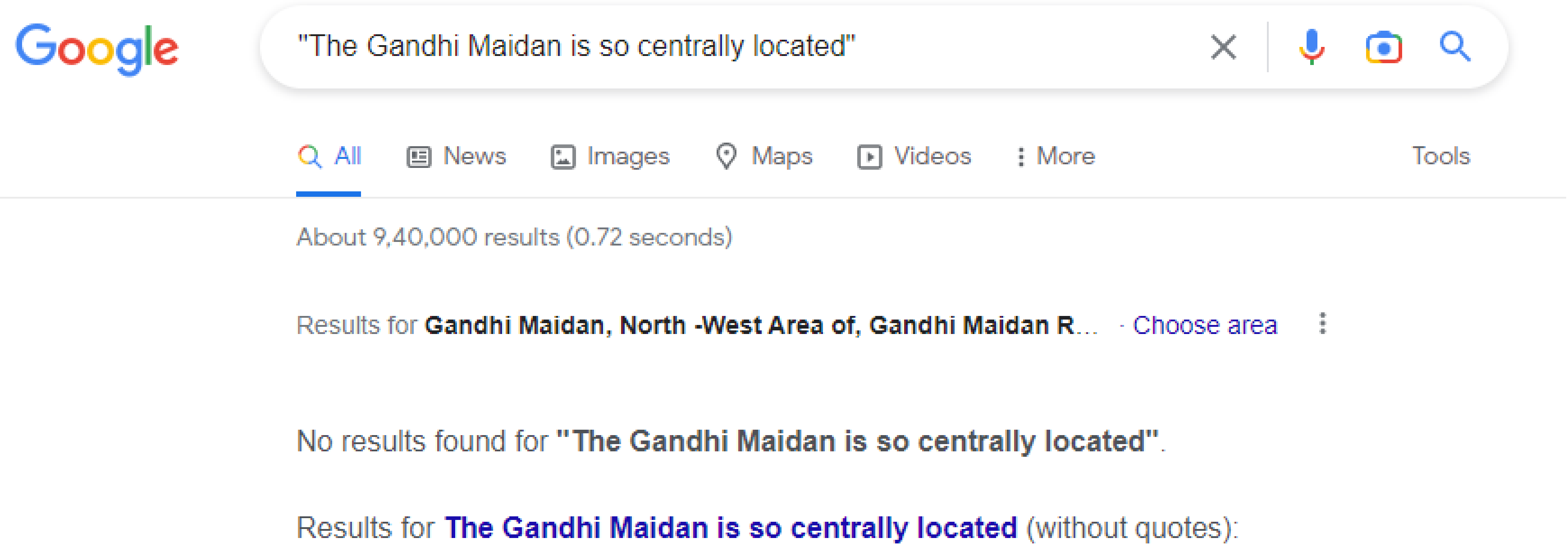
In general , this is an insufficient model of lang because lang has long distance dependencies

- $P(w|h)$
  - Through relative frequency counts: $P(w|h) = \frac{C(wh)}{C(h)}$
  - Let h be "I am so mad at him" and w be "that"
    - $P(w|h) = \frac{C(\text{"I am so mad at him that"})}{C(\text{"I am so mad at him"})}$





- Language is so creative that even the web can fail to provide an estimate



- Solution: approximate using bigram model

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

  - Markov assumption
  - Trigram

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-2:n-1})$$

  - N-gram

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

  - Probability of a sentence

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

IMP

write on desk

- Estimating $P(w_n|w_{n-1})$ throug Maximum Likelihood

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$= \frac{\text{Count of bigram } w_{n-1}w_n}{\text{Count of all bigrams starting with } w_{n-1}}$$

$$= \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

<s> I am Gopal</s>

<s> Indeed, I am happy</s>

<s> I do not like green vegetables</s>

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

$$P(\mathrm{Gopal}| </s >) =$$

$$P(\mathrm{do}|I) =$$

- The general case

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\,w_n)}{C(w_{n-N+1:n-1})}$$

[n-N+1 → n-1]

Ngram

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

Bigram probabilities

	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

From Berkeley Restaurant Project corpus of 9332 sentences followed by

word

Given:

	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

$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25$ $\quad$ $P(\texttt{english}\,|\,\texttt{want}) = 0.0011$

$P(\texttt{food}\,|\,\texttt{english}) = 0.5$ $\quad$ $P(\texttt{</s>}\,|\,\texttt{food}) = 0.68$

Compute the probability of “I want English food” and “I want Chinese food”

Evaluating language models

Extrinsic Evaluation

- Embed an LM in a real-life application and measure overall improvement SPELLING CORRECTION , SPEECH RECOGNIZER
- Expensive
  - Time and resource consuming
- Intrinsic evaluation
  - Internal, independent of any application
  - Train test and test set
    - Dev (test) set
  - Metric: Perplexity

Perplexity (PP)

- Inverse probability of the test set, normalized by the number of words
  - Let $W = w_1, w_2, \ldots, w_N$ be the test set

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

  - minimizing perplexity => maximizing the test set probability.
  - Using bigram model

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

write on desk

  - Formula for trigram and n-gram models?

Perplexity as branching factor

- The branching factor of a language is the number of possible next words that can follow any word.
- Consider a corpus consisting on only digits that are equally distributed
  - $P(\text{a digit}) = \frac{1}{10}$
  - Perplexity = 10 (how?)
  - Perplexity: **weighted** average branching factor
    - Why weighted?

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

$$\left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}}$$

- Wall Street Journal, dataset having a 19,979 word vocabulary.
  - Test set of 1.5 million words

	Unigram	Bigram	Trigram
Perplexity	962	170	109

lower you get the higher is the test set probability

- The more information the n-gram gives us about the word sequence, the lower the perplexity

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

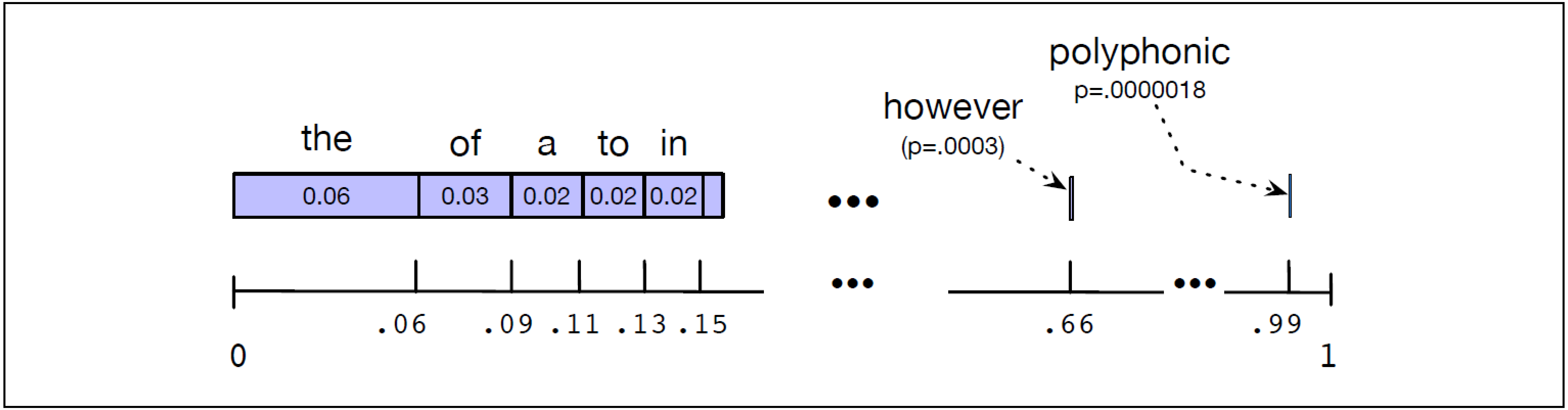
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

MIN PERPLEXITY -> GOOD MODEL

Sampling sentences from a language model

Sampling sentences from a LM

- To generate some sentences, choosing each sentence according to its likelihood as defined by the model.
- Unigram model: keep sampling randomly till </s>



- Can be done with bigram, trigram, …. N-gram

Unigram vs. bigram vs. trigram vs. n-gram

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have not so good sentences –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
quality of sentences increases	
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Generated from Shakespeare's works.

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Generated from Wall Street Journal's dataset.

- Statistical models are likely to be pretty useless as predictors if the training sets and the test sets are as different as Shakespeare and WSJ.
  - Training and test corpus should match w.r.t. genre, dialect, variety etc.

Zero-probability n-grams

- Consider the words that follow the bigram "denied the" in the WSJ Treebank3 corpus

$$\left.\begin{array}{ll}\text{denied the allegations:} & 5\\ \text{denied the speculation:} & 2\\ \text{denied the rumors:} & 1\\ \text{denied the report:} & 1\end{array}\right\}\text{training}$$

- But what if our test has

  denied the offer

  denied the loan

  - Our model will estimate $P(\text{offer}|\text{denied the}) = 0$
- Problem with zeroes
  - Underestimation of probabilities
  - Entire probability of the sentence is zero; perplexity cannot be computed

Out of Vocabulary (OOV) Words

- Add pseudo token <UNK> (unknown word)
- Method 1:
  - Choose a fixed vocabulary in advance
  - Convert OOV words in training set to <UNK> tokens and estimate their probability
- Method 2:
  - Create a vocabulary during training
  - Replace less frequent words with <UNK>

Smoothing

- A way to deal with unknown words
- Shift some probability from frequent words to unknown words
- Techniques
  - Laplace (add-one) smoothing used in text classification
  - Add-k smoothing fraction laplace
  - Backoff and interpolation
  - Kneser-Ney smoothing

Laplace (add-one) smoothing

- Unigram case

$$P(w_i) = \frac{c_i}{N}$$

Where, c_i: count of w_i and N= #word tokens

Becomes

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Where, V: vocab size (#unique tokens)

- Does not perform very well with modern n-gram models write on desk

- $P(w_i) = \frac{c_i}{\boxed{N}}$; $P_{\text{Laplace}}(w_i) = \frac{c_i+1}{\boxed{N+V}}$
- How to capture the effect of smoothing on counts?
  - Adjusted count $c^* = (c_i + 1)\left(\frac{\boxed{N}}{\boxed{N+V}}\right)$
    - Can be converted to a probability by normalizing with N
  - Discounting (lowering) factor:

$$d_c = \frac{c^*}{c} = \frac{(c_i + 1)}{c_i}\left(\frac{N}{N+V}\right)$$

- Bigram case

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\boxed{C(w_{n-1})}}$$

Becomes

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{\boxed{C(w_{n-1})+V}}$$

  - Adjusted count

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times \boxed{C(w_{n-1})}}{\boxed{C(w_{n-1})+V}}$$

write on desk

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

Bigram counts +1

	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

From Berkeley Restaurant Project corpus of 9332 sentences, $|V| = 1446$

Bigram counts +1

	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

Add-one smoothed bigram probabilities
$|V|$ = *1446*

	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

final formula

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

Bigram counts +1

	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

Adjusted counts

	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

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

Bigram probabilities

	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

Add-one smoothed bigram probabilities
$|V| = 1446$

	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

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

Add-k smoothing

- Problem: Add-one smoothing moves too much probability mass to all the zeros.
- Solution: instead of 1, add a fractional count k (.5? .05? .01?)

$$P^{*}_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

- The value of k can be optimized on a ***dev-set***.

Backoff and Interpolation

- Also meant for zero frequency n-grams
  - Estimate n-gram probability using $< n$-grams
- Backoff: use the next lower n-gram with sufficient evidence
- Interpolation: mix and use all lower n-gram probabilities

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

simple interpolation

$\sum_i \lambda_i = 1$ (To make above a probability)

- Interpolation with context-conditioned weights here lambda depends on last words

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2:n-1})P(w_n) + \lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1})$$

- λs are estimated on a separate 'held-out' corpus
  - Choose λs that maximize likelihood of the 'held-out' corpus

Kneser-Ney Smoothing

- One of the most commonly used and best performing n-gram smoothing methods (Kneser and Ney 1995, Chen and Goodman 1998).
- Estimates an **absolute** discounting factor from another "heldout set"
  - Subtracting a fixed (absolute) discount d from each count.

write on desk

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\, v)} + \lambda(w_{i-1})P(w_i)$$

discounted biagram

interpolation weight

unigram

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

$d = .75$