Natural Language Processing

Chapter 2 N-gram Language Model

Dr Raymond Lee
Associate Professor, DST
BNU-HKBU United International College

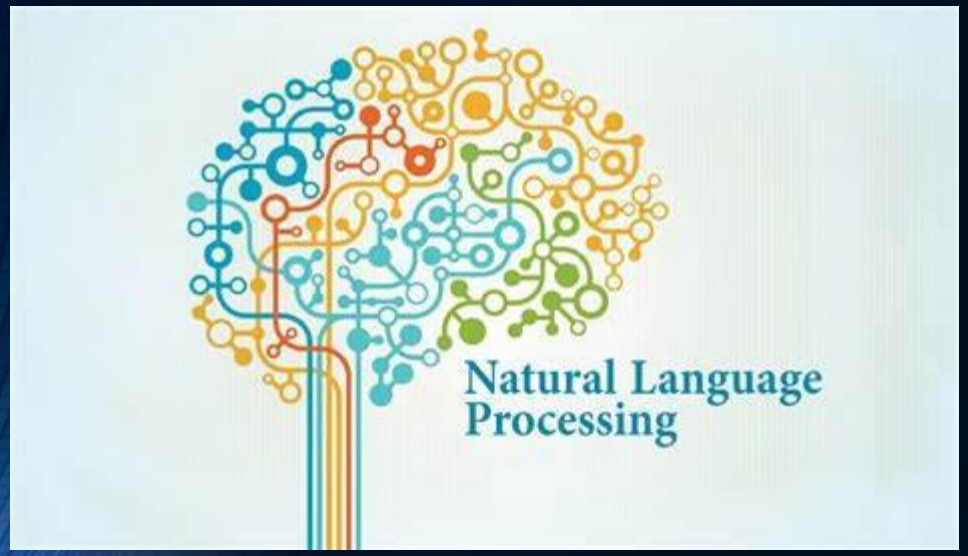
N-gram Language Model

- Introduction and motivation
- N-gram Language Model – The Basics
- Markov Chains in N-gram Language Model
- Live Example – N-Gram Model in The Adventures of Sherlock Holmes
- Shannon's Method in Language Model
- Language Model Evaluation and Smoothing Techniques

PART I

Introduction

From Words to Sequences of Words



- In the first NLP workshop, we are working on word tokenization using NLTK and SpaCy technology.
- Up to this point we've mostly been discussing words in isolation
- Now we're switching to sequences of words
- And we're going to worry about assigning probabilities to sequences of words

Who Cares?

Fig. 2.1 Speech Recognition

- Why would you want to assign a probability to a sentence or…
- Why would you want to predict the next word…
- Lots of applications
  - Speech Recognition
  - Text Classification
  - Text Generation
  - Machine Translation
  - QA Chatbots
  - Auto-driving
  - …

Real-Word Spelling Errors

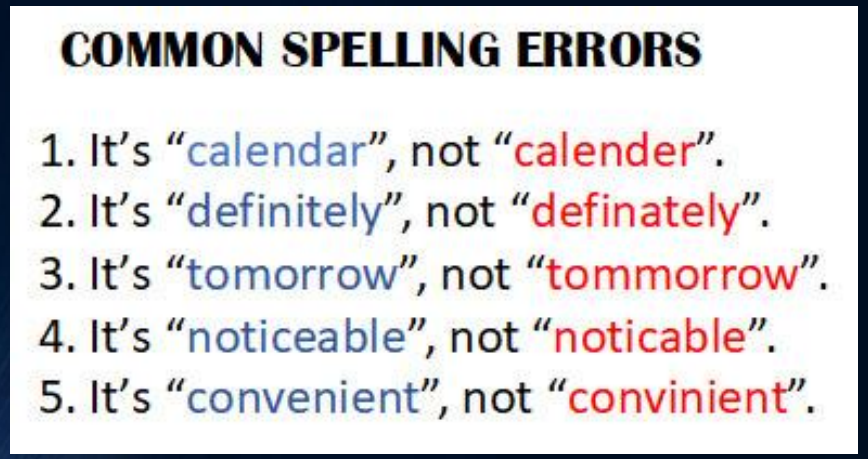


Fig. 2.2 Spelling Errors

- Mental confusions
  - Their/they're/there
  - To/too/two
  - Weather/whether
  - Peace/piece
  - You're/your
- Typos that result in real words
  - Thet for That
  - Lave for Have
  - Sin for Since

Real Word Spelling Errors

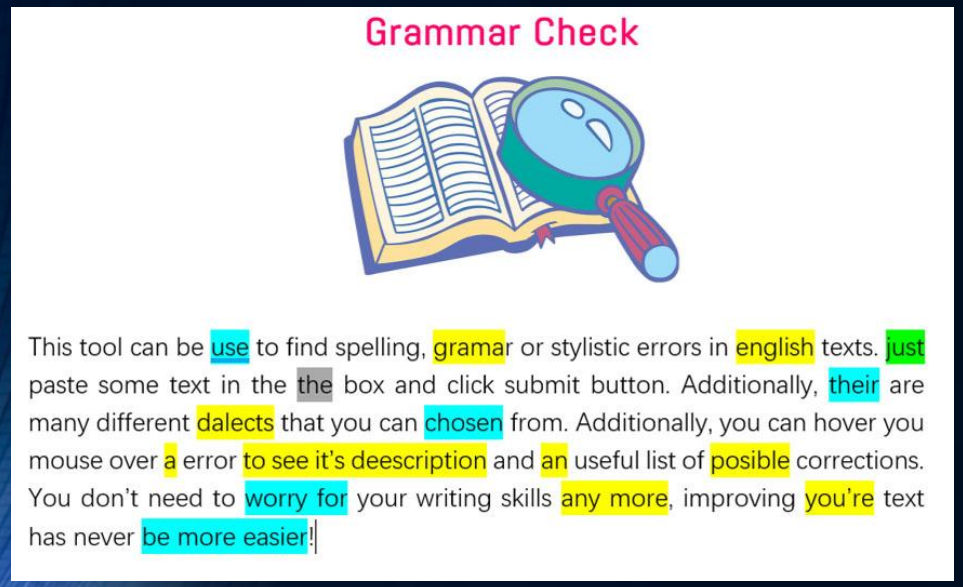


Fig. 2.3 Spelling and grammar checking tools

- Collect a set of common pairs of confusions
- Whenever a member of this set is encountered compute the probability of the sentence in which it appears
- Substitute the other possibilities and compute the probability of the resulting sentence
- Choose the higher one

Next Word Prediction

- Assignment probability of the appearance of the word sequence in a sentence either from a book, corpus, search sequences or chat messages.

Example: Sentence start with "I …"

- I
- I like (0.67%) or I love (0.33%)
- I like Photography (0.67% x 0.5%)
- I like Science (0.67% x 0.5%)
- I love Mathematics (0.33% x 1)
- …

Note: Such probabilities are domain specific.

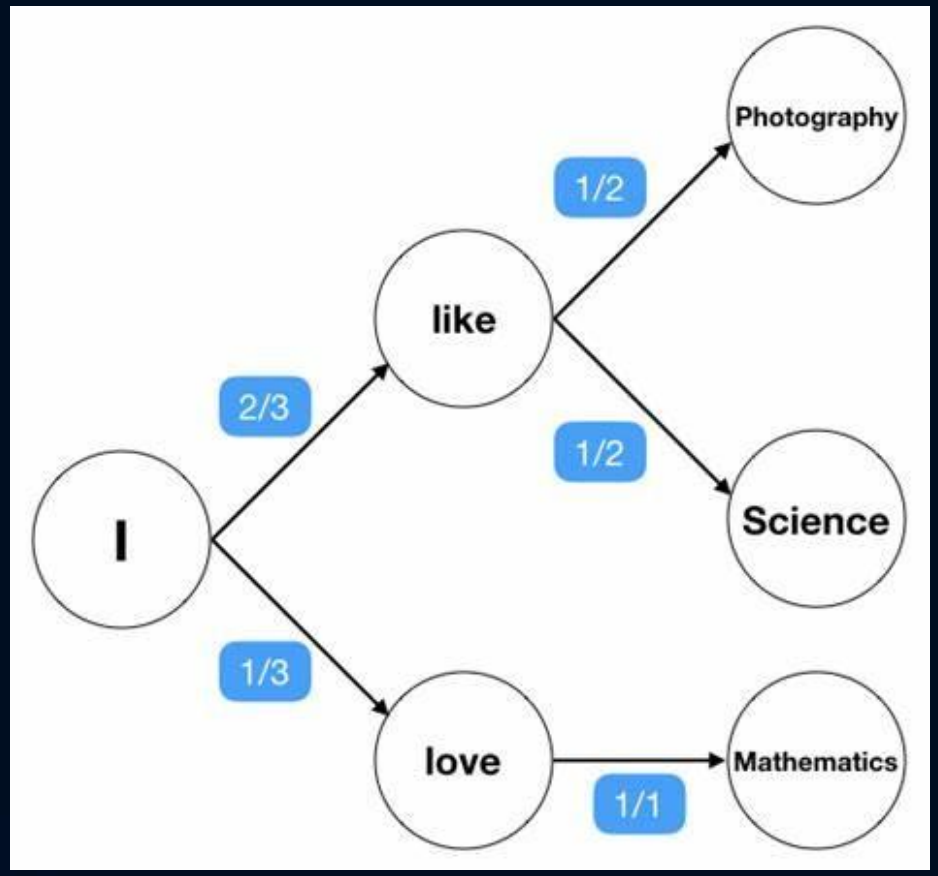


Fig. 2.4 Next word prediction

Human Word Prediction

- Clearly, at least some of us have the ability to predict future words in an utterance.
- How?
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge
  - Common Sense (World Knowledge)

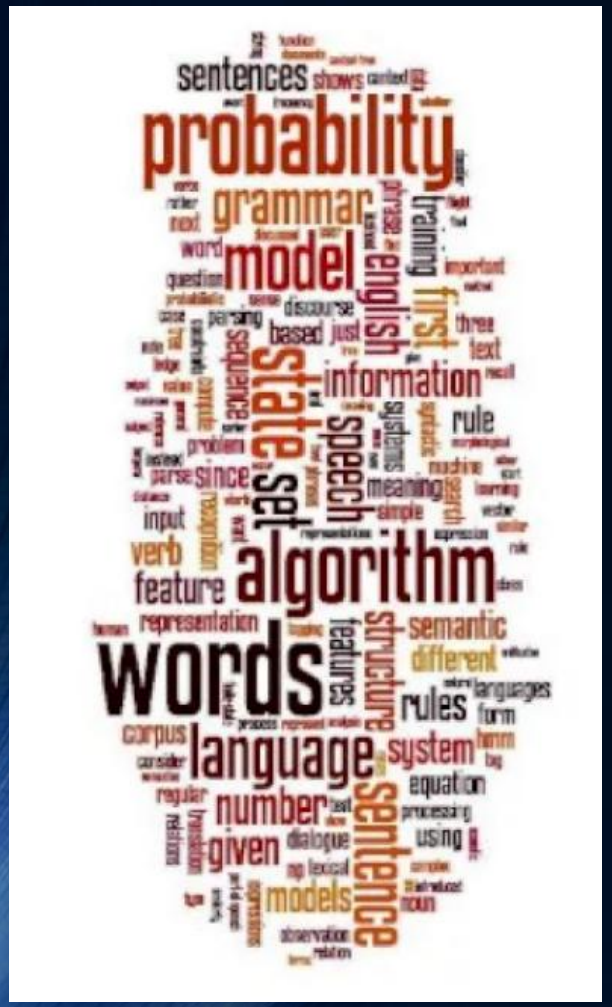
Word Prediction

- Guess the next word...
  - *... I notice three children standing on the ???*
  - *… I just bought some oranges from the ???*
  - *… She stop the car and then open the ???*
- There are many sources of knowledge that can be used to inform this task, either come from 1) Domain knowledge ; 2) World knowledge (Common-sense); or 3) Scenario knowledge.
- But it turns out that you can do pretty well by simply looking at the preceding words and keeping track of some fairly simple counts.
- Let's start with simple word(s) counting method in NLP – Ngram Language Model.

PART II

N-Gram Model

Word Prediction with N-Gram Model



- We can formalize this task using what are called *N*-gram Language Model or N-gram model in short.
- *N*-grams are token sequences of length *N*.
- The most commonly used N-gram model contains 2 words, so called 2-grams Model (aka bigrams)
  - (I notice), (notice three), (three children), (children standing), (standing on), (on the)
- N-gram with N=1 is called Unigram, not commonly used (Why?) but important for the calculation of bigram probability.
- Other commonly used N-grams is Tri-grams, i.e. 3 words sequence.
  - (I notice three), (notice three children), (children standing on), (standing on the), …
- Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.
- The question is: Is it the larger N the better?

N-Gram Models

Conditional Probability Formula

$$\text{Conditional Probability } P(A \mid B) = \frac{P(A \text{ and } B)}{P(B)}$$

- Intuition speaking, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence.
  - E.g. It is not difficult _ ? “that” or “to” or …
  - E.g. I don’t think you are _? “wrong” or “right” or else …
- Or, we can use them to assess the probability of an entire sequence of words.
  - Pretty much the same thing we are all doing in daily conversation …

Applications of N-grams

Fig 2.5 Computerized Axial Tomography Scanner ("Cat scan")

- Why do we want to predict a word, given some preceding words?
  - Rank the likelihood of sequences containing various alternative hypotheses, e.g. ASR (Automatic Speech Recognition)

E.g. The cinema staff told me that the popcorn/unicorn sales have doubled...

  - Assess the likelihood/goodness of a sentence, e.g. for text generation or machine translation

E.g. The doctor recommended a cat scan to the patient.

N-Gram Models of Language

- Use the previous N-1 words in a sequence to predict the next word
- Language Model (LM)
  - unigrams, bigrams, trigrams, quadrigram …
- How do we train these models?
  - Very large corpora such as: knowledgebase and website, public speeches, literature, chat messages, etc.
- Note: different from dictionary, most of them are domain specific
- Is it the larger N-gram the better?

Counting Words in Corpora

- What is a word?
  - e.g., are cat and cats the same word?
  - September and Sept?
  - zero and o, are they the same?
  - Is _ a word? * ? '(' ?
  - How many words are there "$3.14"?
  - How about Chinese ?
    How do we identify a word?

Basic NLP Terminology

Word	Stemming	Lemmatization
information	inform	information
informative	inform	information
computers	comput	computer
feet	feet	foot

Fig 2.6 Stemming vs Lemmatization

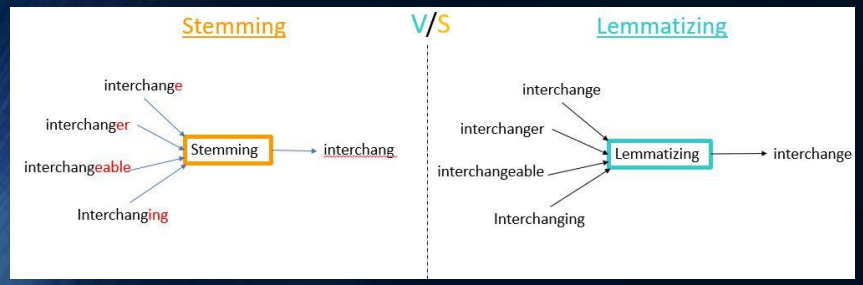
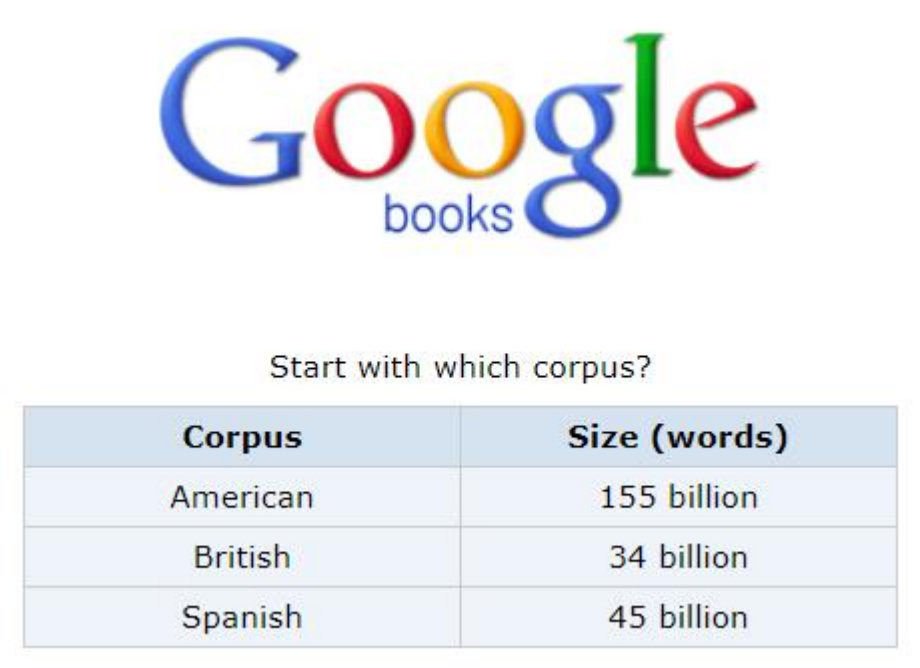


Fig 2.7 Stemming and Lemmatization of word "Interchange"

- Sentence: unit of written language
- Utterance: unit of spoken language
- Word Form: the inflected form that appears in the corpus
- Types: distinct words in a corpus (vocabulary size)
- Tokens: chunk of words
- Stem: the "root" form of the words, stemming is the process of reducing inflected (or sometimes derived) words to their word stem
- Lemma: an abstract form, shared by word forms having the same stem, part of speech, and word sense. Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, which is identified by the word's lemma, or dictionary form.

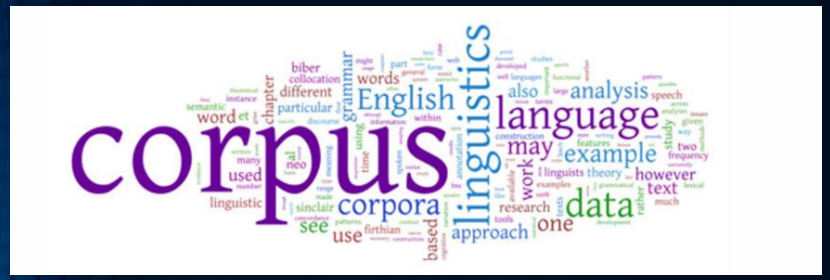
Counting: Corpora



- So what happens when we look at large bodies of text instead of single utterances?
- Brown et al (1992) large corpus of English text
  - 583 million wordform tokens
  - 293,181 wordform types
- Google
  - Crawl of 1,024,908,267,229 English tokens
  - 13,588,391 wordform types
    - That seems like a lot of types... After all, even large dictionaries of English have only around 500k types. Why so many here?

    - Numbers
    - Misspellings
    - Names
    - Acronyms
    - etc

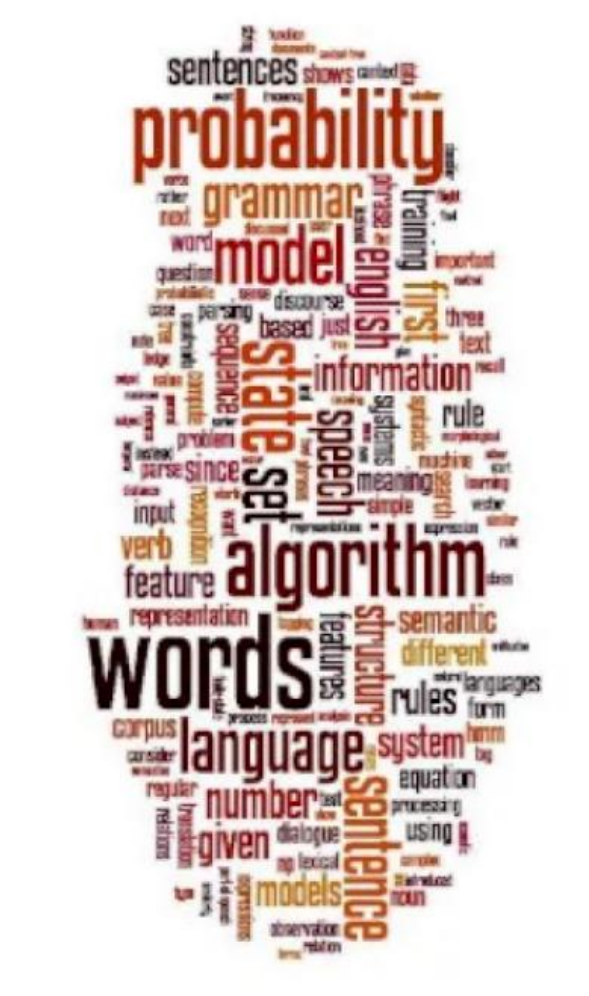
Corpora



https://www.ldc.upenn.edu/

- Corpora are online collections of text and speech
  - Brown Corpus
  - Wall Street Journal
  - AP news
  - Corpus of British Parliament speeches (Hansard)
  - The Boston university radio news corpus
  - NLTK Corpora Library

Language Modeling



- Back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence
  - $P(w_n|w_1,w_2 \ldots w_{n-1})$
- E.g. Guess the utterance "It is difficult to say/tell/guess that …"
- Traditionally, we'll call a statistical model that can assess this a *Language Model*
- *Nowadays, with the advance of AI and Deep network technology, language model can be based on Deep Networks and Recurrent Neural Networks (RNNs).*

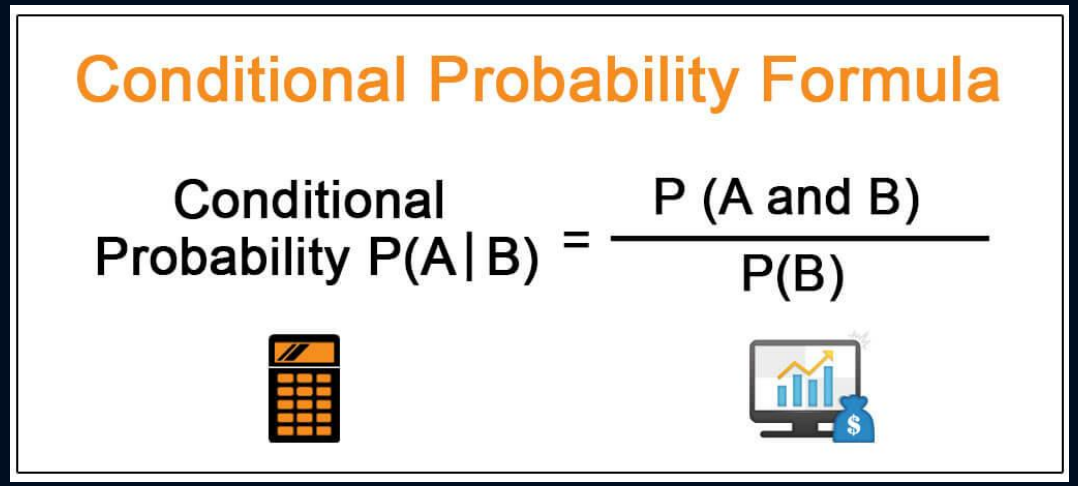
Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts.
    So to get
  - P(*that* | *The garden is so beautiful*)
- By definition that's

  P(The garden is so beautiful that)

  P(The garden is so beautiful)

  We can get each of those from counts in a large corpus.



Very Easy Estimate

- How to estimate?
  - P(that | the garden is so beautiful)

P(that | The garden is so beautiful) =

$$\frac{\text{Count(The garden is so beautiful that)}}{\text{Count(The garden is so beautiful)}}$$

Language Modeling

Conditional Probability

4. Bayes' Theorem:

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)} = \frac{P(F \mid E)P(E)}{P(F)}$$

5. Chain Rule

$$P(E_1 \cdots E_n)$$

$$= P(E_1)P(E_2 \mid E_1)P(E_3 \mid E_1 E_2) \cdots P(E_n \mid E_1 \cdots E_{n-1}).$$

9

- Unfortunately, for most sequences and for most text collections we won't get good estimates from this method.
  - What we're likely to get is 0. Or worse 0/0.
- Clearly, we'll have to be a little more clever.
  - Let's use the chain rule of probability
  - And a particularly useful independence assumption.

The Chain Rule

- Recall the definition of conditional probabilities

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

- Rewriting:

$$P(A \wedge B) = P(A|B)P(B)$$

- For sequences…
  - P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)
- In general
  - $P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n-1})$

The Chain Rule

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

P(the garden is so beautiful that)=

P(the)*

P(garden | the)*

P(is | the garden)*

P(so | the garden is)*

P(beautiful| the garden is so)

P(that | the garden is so beautiful)

Note: For better formulation we use <s> and </s> to denote the start and end of the sentence (utterance).

i.e. <s>the garden is so beautiful that</s>

Word Sequence Probability

- The word sequence from position 1 to n is dented w_1^n
- So the probability of a sequence is

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1})$$
$$= P(w_1)\prod_{k=2}^{n} P(w_k \mid w_1^{k-1})$$

Unfortunately

Two major problems

- That doesn't help since its unlikely we'll ever gather the right statistics for the prefixes
- The calculation of the conditional probability of the word sequence is very tedious and most of the time is rather difficult to achieve

PART III

Markov Chains in N-gram Language Model

Markov Assumption in Markov Chain Theory

Andrei Andreyevich Markov (1856 – 1922) is a well-known Russian mathematician, academician who made a great contribution to science, studying the theory of probability. A primary subject of his research later became known as Markov chains or Markov processes, which have been applied in various areas such as: Biology, Chemistry, Computer Science and Statistics.
In terms of Computer Science, Markov Chain Theory can be applied to Speech Recognition, N-gram Model, Internet Ranking, Information Theory and Queueing theory.
-> e.g. Character and voice recognition.

Fig 2.8 Andrei Andreyevich Markov
(1856 - 1922)

Markov Assumption

- Assume that the entire prefix history isn't necessary.
- In other words, an event doesn't depend on all of its history, just a fixed length near history
- → Markov Chain Theory

Markov Chain

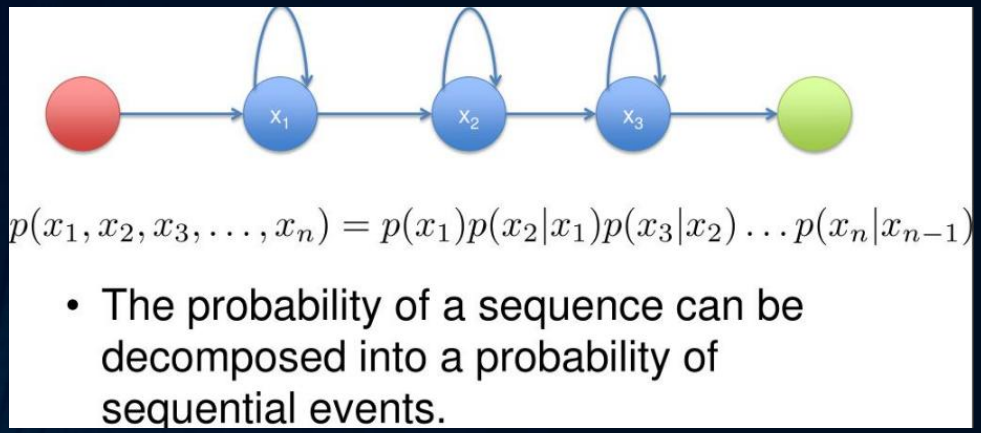


Fig 2.9 Markov Chain

Internet applications [edit]

The PageRank of a webpage as used by Google is defined by a Markov chain.[80][81][82] It is the probability to be at page i in the stationary distribution on the following Markov chain on all (known) webpages. If N is the number of known webpages, and a page i has k_i links to it then it has transition probability $\frac{\alpha}{k_i} + \frac{1-\alpha}{N}$ for all pages that are linked to and $\frac{1-\alpha}{N}$ for all pages that are not linked to. The parameter α is taken to be about 0.15.[83]

Markov models have also been used to analyze web navigation behavior of users. A user's web link transition on a particular website can be modeled using first- or second-order Markov models and can be used to make predictions regarding future navigation and to personalize the web page for an individual user.

Markov chain

From Wikipedia, the free encyclopedia

A **Markov chain** or **Markov process** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.[1][2][3] A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain (DTMC). A continuous-time process is called a continuous-time Markov chain (CTMC). It is named after the Russian mathematician Andrey Markov.

Markov chains have many applications as statistical models of real-world processes,[1][4][5][6] such as studying cruise control systems in motor vehicles, queues or lines of customers arriving at an airport, currency exchange rates and animal population dynamics.[7]

Markov processes are the basis for general stochastic simulation methods known as Markov chain Monte Carlo, which are used for simulating sampling from complex probability distributions, and have found application in Bayesian statistics, thermodynamics, statistical mechanics, physics, chemistry, economics, finance, signal processing, information theory and speech processing.[7][8][9]

Markov Chain in NLP

- So for each component in the product replace each with the approximation (assuming a prefix of N)

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

N-Grams
Example: The big white cat

- Unigrams: P(cat)
- Bigrams: P(cat | white)
- Trigrams: P(cat | big white)
- Quadrigrams: P(cat | the big white)

In general, we'll be dealing with

P(Word | Some fixed prefix)

Warning!

- The formulation P(Word| Some fixed prefix) might not really appropriate in many applications.
- It is if we're dealing with real time speech where we only have access to prefixes.
- But if we're dealing with text we already have the right and left contexts. There's no a priori reason to stick to left contexts.

PART IV

Live Example – N-Gram Model in The Adventures of Sherlock Holmes

Training and Testing

- N-Gram probabilities come from a training corpus
  - overly narrow corpus: probabilities don't generalize
  - overly general corpus: probabilities don't reflect task or domain
- A separate test corpus is used to evaluate the model, typically using standard metrics
  - held out test set; development test set
  - cross validation
  - results tested for statistical significance

A Simple Example from “Adventures of Sherlock Holmes” by Sir Arthur Canon Doyle

The Adventures of Sherlock Holmes contains 12 famous stores which can be free accessed and download from Project Gutenberg website https://www.gutenberg.org/

1. A Scandal in Bohemia
2. The Red-Headed League
3. A Case of Identity
4. The Boscombe Valley Mystery
5. The Five Orange Pips
6. The Man with the Twisted Lip
7. The Adventure of the Blue Carbuncle
8. The Adventure of the Speckled Band
9. The Adventure of the Engineer’s Thumb
10. The Adventure of the Noble Bachelor
11. The Adventure of the Beryl Coronet
12. The Adventure of the Copper Beeches

No. of pages: 424
No. of characters: 470,119 (exclude spaces)
No. of words: 110,087
No. of token: 113,749
No. of sentence: 6830
No. of word types (V): 9886

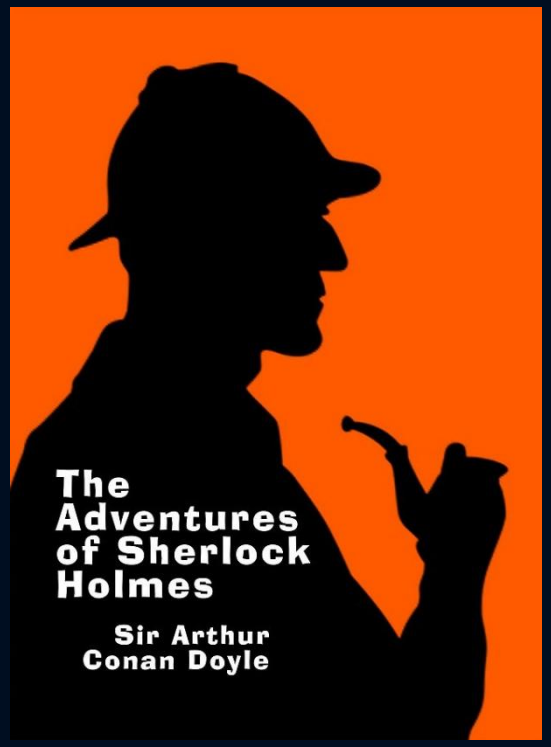


Fig 2.10 The Adventures of Sherlock Holmes

His famous quote: “I have no doubt that I …”

What is the N-gram probability using Markov Chain Method?

P(I have no doubt that I)

= P(I | <start>) P(have | I) P(no | have) P(doubt | no) P(that | doubt) P(I | that)

A Bigram Grammar Fragment of Adventures of Sherlock Holmes

I	have	no	doubt	that
2755	867	276	84	1767

Fig 2.11 Unigram counts for words in "I have no doubt that"

For example: Given the unigram count of word "I" is 2755. The conditional probability of "I have" using Markov Chain Method will be: P(have | I) = 288/2755 = 0.105.

Bigram with "I" (By counts)

I have	288	I observe	8
I had	161	I deduce	3
I am	159	I can	37
I was	147	I can't	6
I knew	34	I may	23
I hear	33	I must	32
I don't	14	I could	77
I saw	42	I passed	8
I think	72	I take	4
I should	90	I see	32

Bigram with "I" (By probability)

I have	0.105	I observe	0.003
I had	0.058	I deduce	0.001
I am	0.058	I can	0.013
I was	0.053	I can't	0.002
I knew	0.012	I may	0.008
I hear	0.012	I must	0.012
I don't	0.005	I could	0.028
I saw	0.015	I passed	0.003
I think	0.026	I take	0.001
I should	0.033	I merely	0.012

Fig 2.12 Bigram Grammar Fragment from "The Adventures of Sherlock Holmes"

A Bigram Grammar Fragment of Adventures of Sherlock Holmes Bigram related to "I have no doubt that I" (By counts)

Bigram related to "I have no doubt that I" (By counts)

<s>I	883	no doubt	46
<s>I'd	4	no sign	9
<s>The	164	no harm	4
<s>It	229	doubt that	17
I have	288	doubt as	3
I had	161	doubt upon	2
I can	37	that I	228
have no	35	that he	139
have to	12	that she	61
have been	122	that it	109

Bigram related to "I have no doubt that I" (By probability)

<s>I	0.138	no doubt	0.167
<s>I'd	0.001	no sign	0.033
<s>The	0.026	no harm	0.014
<s>It	0.036	doubt that	0.202
I have	0.105	doubt as	0.036
I had	0.058	doubt upon	0.024
I can	0.013	that I	0.129
have no	0.040	that he	0.079
have to	0.014	that she	0.035
have been	0.141	that it	0.062

Fig 2.13 Bigram Grammar Fragment from "The Adventures of Sherlock Holmes" related to the utterance "I have no doubt that"

Try the N-gram Probabilities

- P(I have no doubt that I)
  = P(I | <s>) P(have | I) P(no | have) P(doubt | no) P(that | doubt) P(I | that)
  = 0.138 x 0.105 x 0.040 x 0.167 x 0.202 x 0.129
  = 0.000002526
- vs. P(I have no doubt that he)
  = P(I | <s>) P(have | I) P(no | have) P(doubt | no) P(that | doubt) P(I | he)
  = 0.138 x 0.105 x 0.040 x 0.167 x 0.202 x 0.079
  = 0.000001540

Exercise: Try to calculate the P(I have no doubt that she) and P(I have no doubt that it) and compare their probabilities.

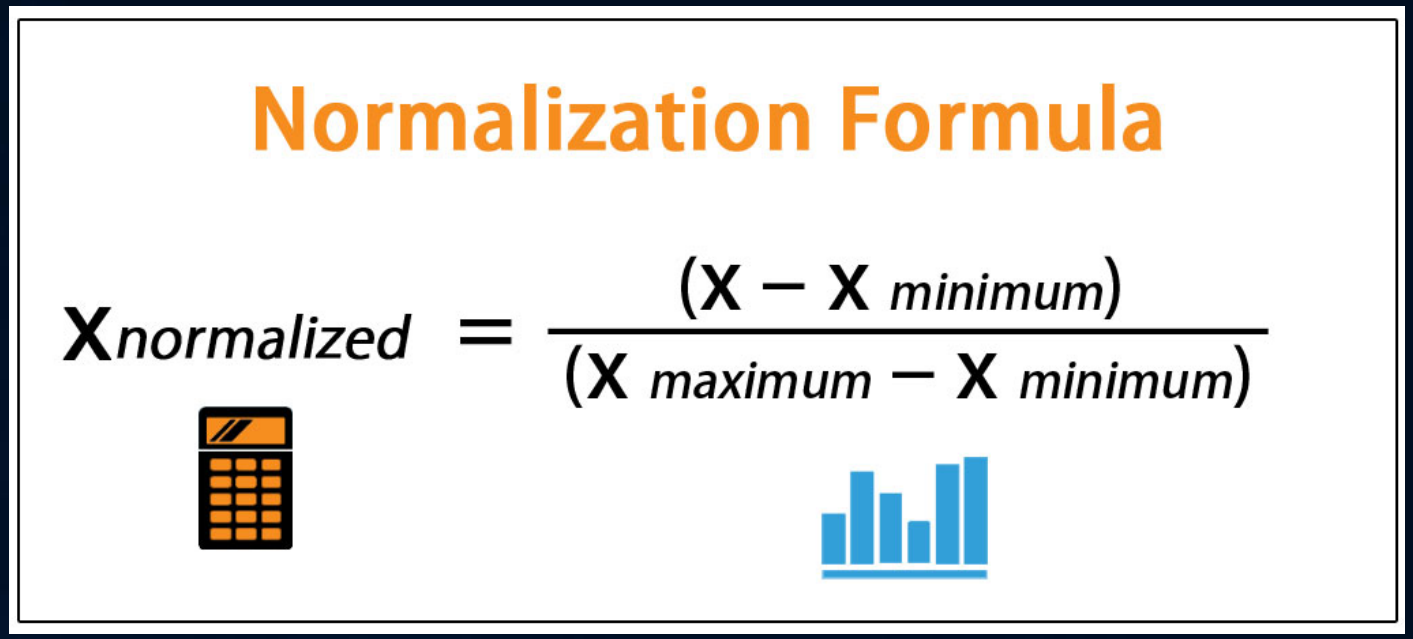
- Probabilities seem to capture both ``syntactic'' facts and ``world knowledge''
  - Although “that I” or “that he” is commonly used in English grammar, probably in Sherlock Holmes story, using “that I” is more frequent! (Why?)
- As seen, most of the time the N-gram probability of a sentence (utterance) is always very small. Why?
- The truth is, N-gram models can be trained by counting and normalization.

Some Observations

- You don't really do all those multiplies. The numbers are too small and lead to underflows
- Convert the probabilities to logs and then do additions.
- To get the real probability (if you need it) go back to the antilog.

How do we get the N-gram probabilities?

- N-gram models can be trained by counting and normalization



Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

Maximum Likelihood Estimates (MLE)

- The maximum likelihood estimate of some parameter of a model M from a training set T
  - Is the estimate that maximizes the likelihood of the training set T given the model M
- Suppose the word “cloud” occurs 350 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be “cloud”
- MLE estimate is 350/1000000 = 0.00035
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that the word “cloud” will occur 350 times in a million word corpus like Brown Corpus.

Bigram Counts
Come back to the Adventures of Sherlock Holmes example

	I	have	no	doubt	that	I
I	0	288	0	1	0	0
have	5	0	35	0	2	5
no	0	0	0	46	0	0
doubt	0	0	0	0	17	0
that	228	1	10	0	7	228

Fig 2.14 Bigram Counts for "I have no doubt that I" in Adventures of Sherlock Holmes

BERP Bigram Probabilities

- Bigram Normalization: divide each row's counts by appropriate unigram counts for w_{n-1}

I	have	no	doubt	that
2755	867	276	84	1767

Fig 2.11 Unigram counts for words in "I have no doubt that"

- Example: Computing the bigram probability of "no doubt"
  - C(doubt, no)/C(no)
  - p (doubt | no) = 46 / 276 = 0.167 which is much higher than P(have | I) = 0.105 which is rather unique and different from other corpus.
- Maximum Likelihood Estimation (MLE): relative frequency of e.g.

$$\frac{freq(w1,w2)}{freq(w1)}$$

The overall Bigram Probability (normalized)

	I	have	no	doubt	that	I
I	0.000	0.105	0.000	0.000	0.000	0.000
have	0.006	0.000	0.040	0.000	0.002	0.006
no	0.000	0.000	0.000	0.167	0.000	0.000
doubt	0.000	0.000	0.000	0.000	0.202	0.000
that	0.129	0.001	0.006	0.000	0.004	0.129

Fig 2.15 Bigram Probability (Normalized) for "I have no doubt that I" in Adventures of Sherlock Holmes

Some findings:

1. All conditional probabilities are rather small (Why?)
2. Most of them are Zeros
3. Probabilities are domain specific

e.g. P(no doubt) much higher than P(I have), P(have no) or even P(that I) which is commonly used in English literature.

What do we learn about the language?

- What's being captured with ...
  - P(have | I) = 0.105
  - P(no | have) = 0.040
  - P(doubt | np) = 0.167
  - P(that | doubt) = 0.202
  - P(I | that) = 0.129

  => What is that meant in terms of Language Model???
- What about...
  - P(I | I) = 0.000
  - P(have | I) = 0.105
  - P(knew | I) = 0.012
  - P(think | I) = 0.026

What Kinds of Knowledge we can obtain

- As crude as they are, *N*-gram probabilities capture a range of interesting facts about language.

1. Syntax

   E.g. P(have | I) = 0.105

2. World Knowledge

   E.g. P(doubt | no) = 0.167

3. Discourse

   E.g. P(that | no) = 0.000

Which are more useful for US???
→ Text generation …

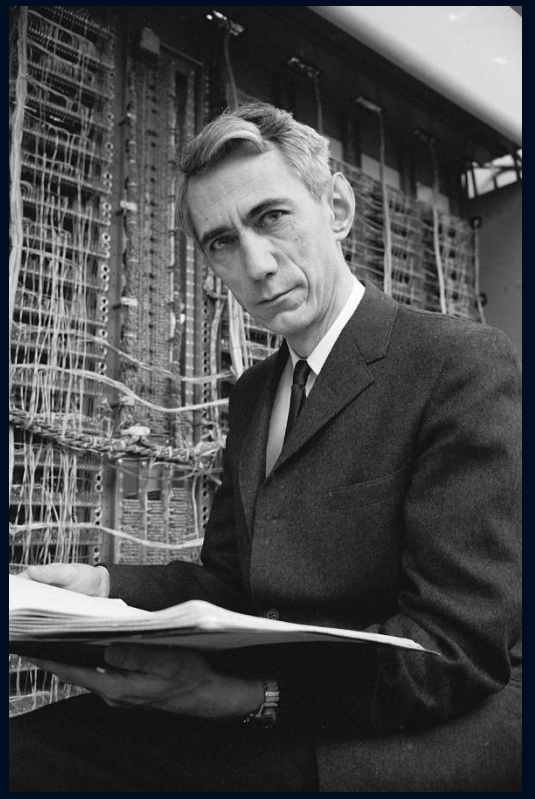
PART V

Shannon's Method in Language Model

Shannon's Method

Claude Elwood Shannon (1916 - 2001) was an American mathematician, electrical engineer, and cryptographer known as a "father of information theory".
As a 21-year-old master's degree student at the Massachusetts Institute of Technology (MIT), he wrote his thesis demonstrating that electrical applications of Boolean algebra could construct any logical numerical relationship. Prof. Shannon contributed to the field of cryptanalysis for national defence of the United States during World War II, including his fundamental work on codebreaking and secure telecommunications.
Shannon's most important paper, ' A mathematical theory of communication ,' was published in 1948. This fundamental treatise both defined a mathematical notion by which information could be quantified and demonstrated that information could be delivered reliably over imperfect communication channels like phone lines or even wireless connections. These ground-breaking innovations provided the tools that provides the foundation of network communication and Internet technology.

- Assigning probabilities to sentences is all well and good, but it's not terribly illuminating .
- A more interesting task is to turn the model around and use it to generate random sentences that are *like* the sentences from which the model was derived.



Prof Claude Shannon (1916 – 2001)

Shannon's Method

- Sample a random bigram (<s>, w) according to its probability
- Now sample a random bigram (w, x) according to its probability
  - Where the prefix w matches the suffix of the first.
- And so on until we randomly choose a (y, </s>)
- Then string the words together

Example:

<s> I

I have

have no

no doubt

doubt that

that I

I …

… </s>

Or "I have no doubt to tell you that …

Or "I have no reason to treat you ..

Or "I have so many thing to tell ..

Shannon's Method on Language Generation

1. Choose a random N-gram (<s>, w) according to its probability
2. Now choose a random N-gram (w, x) according to its probability
3. And so on until we choose </s>
4. Then string the words together into a sentence.

Fig 2.16 Algorithm of Shannon's Method on Language Generation

Try Shakespeare's work

N-gram	Generated Sample Sentences from Shakespeare Works
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. • Hill he late speaks 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? • Why dost stand forth thy canopy for sooth. • What we hath got so she I rest and sent to scold and nature bankrupt nor the first gentlemen? • Ener Menenius if it so many good direction found thou art a strong upon command of fear not a liberal largess given away.
Trigram	• Sweet prince Falstaff shall die. • 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.
Quadri-gram	• King Henry I will go seek the traitor Gloucester. • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long.

Fig 2.18 Sample Sentence Generation using Shannon Method with Shakespeare works

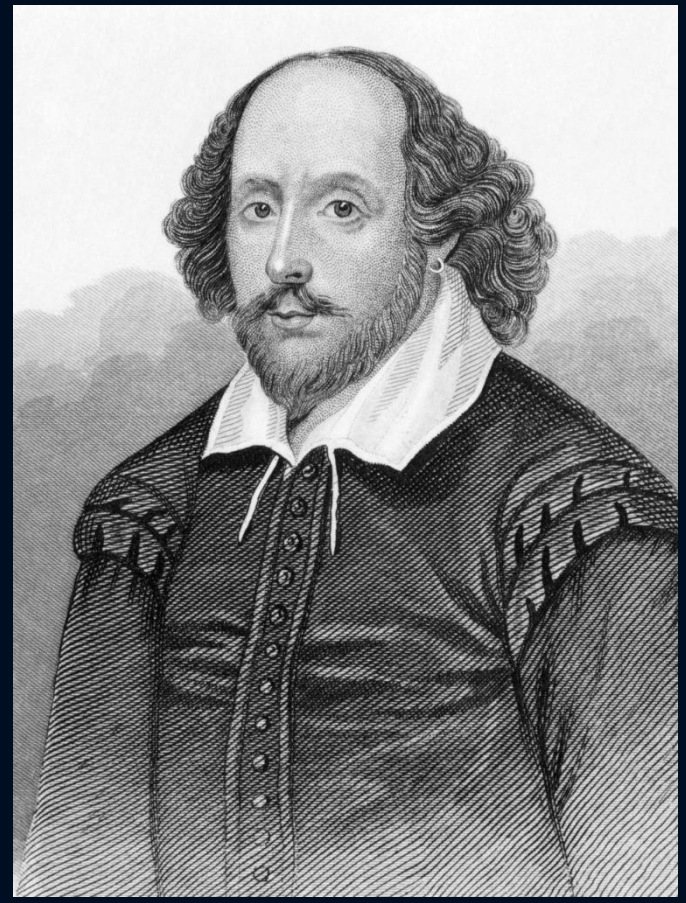


Fig 2.17 Sir William Shakespeare (1564 – 1616)

Shakespeare as a 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)
  - This is the biggest problem in language modeling; we'll come back to it.
- Quadrigrams are worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

How are about : The Wall Street Journal ?

N-gram	Generated Sample Sentences from Wall Street Journal Articles
Unigram	Months the my and issue of year foreign new exchange's September were recession exchange new endorsed a acquire to six executes.
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.
Quadri-gram	Executives from some of the biggest U.S. news organizations check with a British economist last year at Washington's exclusive Metropolitan Club to strategize with a mutual obsession of getting their industry out from under the thumb of Google and Facebook.

Fig 2.19 Sample Sentence Generation using Shannon Method with Wall Street Journal Articles

Evaluation of N-gram Model Performance

- How do we know if our models are any good?
  - And in particular, how do we know if one model is better than another.
- Well Shannon's model gives us an intuition.
  - The generated texts from the higher order models sure look better.
  - That is, they sound more like the text the model was obtained from.
  - But what does that mean?
  - Can we make that notion operational?

PART VI

Language Model Evaluation and Smoothing Techniques

Evaluation

- Standard method
  - Train parameters of our model on a **training set**.
  - Look at the model performance on some new data
    - This is exactly what happens in the real world; we want to know how our model performs on data we haven't seen
  - So, we use a **test set** - A dataset which is different than our training set, but is drawn from the same source
  - Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
    - One such metric is so-called **perplexity** (to be introduced shortly)

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before (pretty much regardless of how much training data you have).
- With an *Open Vocabulary* task
  - Create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L, of size V
      - From a dictionary or
      - A subset of terms from the training set
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we count that like a normal word
  - At test time
    - Use <UNK> counts for any word not in training

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

- For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

Lower perplexity means a better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Fig 2.20 Perplexity values for WSJ from Unigram to Trigram

Evaluating *N*-Gram Models

- Best evaluation for a language model
  - Put model A into an application
    - For example, a speech recognizer or even a QA chatbot
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models
  - ***Extrinsic evaluation***

Difficulty of extrinsic evaluation of N-gram models

- Extrinsic evaluation
  - This is really time-consuming for implementation and system testing
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **just** like the training data
  - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**
  - **But is helpful to think about.**

How to handle : Zero Counts?

- Back to our Adventures of Sherlock Holmes
  - Recall that Shakespeare produced 109,139 bigram types out of over 100 million possible bigrams…
  - So, 99.89% of the possible bigrams were never seen (have zero entries in the table)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?

Zero Counts Problems

- Some of those zeros are really zeros…
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).
- Zipf's Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!
- **Answer:**
  - Estimate the likelihood of unseen (zero count) N-grams!

Smoothing Techniques

- Every n-gram training matrix is sparse, even for very large corpora **(Zipf's law)**
- Solution: estimate the likelihood of unseen n-grams
- Problems: how do you adjust the rest of the corpus to accommodate these 'phantom' n-grams?

Problem

- Let's assume we're using N-grams
- How can we assign a probability to a sequence where one of the component n-grams has a value of zero
- Assume all the words are known and have been seen
  1. Go to a lower order n-gram
  2. Back off from bigrams to unigrams
  3. Replace the zero with something else

Smoothing Techniques

1. Laplace (Add-one) Smoothing
2. Add-k Smoothing
3. Backoff and Interpolation Smoothing
4. Good Turing Smoothing

Laplace (Add-one) Smoothing

- Make the zero counts 1.
- Rationale:
  - They're just events you haven't seen yet.
  - If you had seen them, chances are you would only have seen them once… so make the count equal to 1.

Laplace (Add-one) Smoothing

- For unigrams:
  - Add 1 to every word (type) count
  - Normalize by N (tokens) /(N (tokens) +V (types))
  - Smoothed count (adjusted for additions to N) is:

$$(ci + 1)\frac{N}{N + V}$$

  - Normalize by N to get the new unigram probability:

$$p_i^* = \frac{ci + 1}{N + V}$$

- For bigrams:
  - Add 1 to every bigram $c(w_{n-1} w_n) + 1$
  - Increment the unigram count by vocabulary size $c(w_{n-1}) + V$

Come back to our Sherlock Holmes Example
(Bigram count before and after Laplace Method)

Orginal Bigram Table of "I have no doubt that I"
(By Bigram Count)

	I	have	no	doubt	that	I
I	0	288	0	1	0	0
have	5	0	35	0	2	5
no	0	0	0	46	0	0
doubt	0	0	0	0	17	0
that	228	1	10	0	7	228

Bigram Table of "I have no doubt that I" with Laplace Method
(By Bigram Count)

	I	have	no	doubt	that	I
I	1	289	1	2	1	1
have	6	1	36	1	3	6
no	1	1	1	47	1	1
doubt	1	1	1	1	18	1
that	229	2	11	1	8	229

Fig 2.21 Bigram Counts with and without Laplace Method

Come back to our Sherlock Holmes Example
(Bigram probability before and after Laplace Method)

Bigram Probabilities for normal Bigrams

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

Bigram Probabilities with Laplace Method

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

Note: Originally, the bigram probability is calculated by the division of the unigram, but now will be division by the count of unigram + total number of word type (V) which is equal to 9886.
For example, p(have | I) originally is 288/2755 = 0.105,
using Laplace Method it becomes 288/(2755+9886) = 0.023.

Orginal Bigram Table of "I have no doubt that I"
(By Bigram Probability)

	I	have	no	doubt	that	I
I	0.00000	0.10454	0.00000	0.00036	0.00000	0.00000
have	0.00577	0.00000	0.04037	0.00000	0.00231	0.00577
no	0.00000	0.00000	0.00000	0.16667	0.00000	0.00000
doubt	0.00000	0.00000	0.00000	0.00000	0.20238	0.00000
that	0.12903	0.00057	0.00566	0.00000	0.00396	0.12903

Bigram Table of "I have no doubt that I" with Laplace Method
(By Bigram Probability)

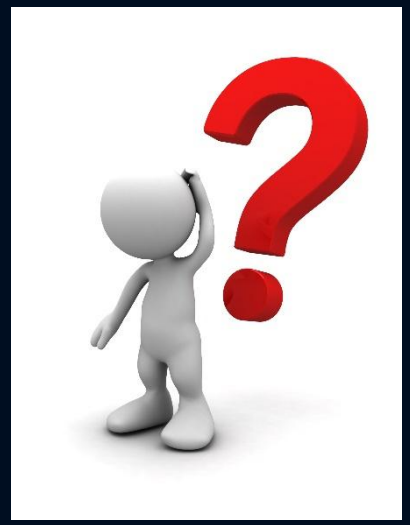
	I	have	no	doubt	that	I
I	0.00008	0.02286	0.00008	0.00016	0.00008	0.00008
have	0.00056	0.00009	0.00335	0.00009	0.00028	0.00056
no	0.00010	0.00010	0.00010	0.00463	0.00010	0.00010
doubt	0.00010	0.00010	0.00010	0.00010	0.00181	0.00010
that	0.01965	0.00017	0.00094	0.00009	0.00069	0.01965

Fig 2.22 Bigram Probabilities with and without Laplace Method

The Problem is …

- Discount: ratio of new counts to old e.g. add-one smoothing changes the Bigram probability of p(have | I) from 0.105 to 0.023 around 4.5 times smaller.

- Problem: add one smoothing changes counts drastically:
  - too much weight given to unseen n-grams
  - in practice, unsmoothed bigrams often work better!



Add-k Smoothing

- Rationale: pretend we've seen each n-gram k times more than we have.
- Instead of adding 1 to each count, we add a non-integer count k (e.g. 0.05, 0.1, 0.2).
- Typically, $0 < k \leq 1$.

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

Note:

Although add-k is useful for some tasks (including text classification), it turns out that it still doesn't work well for language modelling, generating counts with poor variances and often inappropriate discounts (Gale and Church, 1994).

Backoff and Interpolation Smoothing

Rationale:

- If we have no example of a particular N-gram, we look for the N-1 gram. If N-1 gram still haven't got the number count, we switch to N-2 gram, etc.

Method:

- For example, if we have no example of a trigram, and we can instead estimate its probability by using a bigram.
- Similarly, if we don't have a bigram either, we can look up to unigram.
- This is a backoff method and by interpolation, always mix the probability estimates from all the n-gram, weighing and combining the trigram, bigram, and unigram count.
- The Backoff and Interpolation technique we use is we combine different orders of n-grams ranging from say Unigram to 4-grams for the model with simple interpolation method.
- For example, we calculate trigram probability together unigram, bigram, and trigram, each weighted by λs. Noted that the sum of all λs must be 1.

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

Backoff and Interpolation Smoothing

- In a more sophisticated version of linear interpolation, each lambda λ weight can be calculated by conditioning on the context.
- In this way, if we have accurate numbers of a particular bigram, we can assume the number of trigrams based on this bigram, which will be a more robust method to implement so the equation can be:

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

Note:

- Both the simple interpolation and conditional interpolation methods are learned from a held-out corpus.
- A held-out corpus is an additional training corpus that we use to set hyperparameters like these λ values by choosing the λ values that maximize the likelihood of the held-out corpus.
- We adjust the n-gram probabilities and then search for the λ values that give us the highest probability of the held-out set.
- There are numerous approaches to find this optimal set of λ.
- The straightforward way is to use the EM algorithm, an iterative learning algorithm that converges on locally optimal λ (Jelinek and Mercer, 1980)

Good Turing Smoothing

Rationale:

- Use total frequency of events that occur only once to estimate how much mass to shift to unseen events
- Using a bag of color beans as example to estimate the probability of an unseen color bean (e.g. red).

Method:

- This technique uses the frequency of occurring of N-grams reallocates probability distribution using two criteria.
- For example, in our case of N-gram statistics of Adventures of Sherlock Holmes shown in Fig. 2.20.
- As we saw above, P("have doubt") equals 0 without smoothing. We then use the frequency of bigrams that occurred once i.e. P("I doubt"), the total number of bigrams for unknown bigrams.
- $P_{unknown}(w_i \mid w_{(i-1)})$ = (count of bigrams that appeared once) / (count of total bigrams)
- For known bigrams such as "no doubt", we use the frequency of bigrams that occurred more than one of the current bigram frequency (N_{c+1}), frequency of bigrams that occurred the same as the current bigram frequency (N_c), and the total number of bigrams(N).
- $P_{known}(w_i \mid w_{(i-1)}) = c^* / N$
- Where $c^* = (c+1) * (N_{c+1}) / (N_c)$ and c = count of input bigram.

Exercise: Try to calculate these two probabilities from the data provided by Fig. 2.20 as an exercise.

Summary

- N-gram probabilities can be used to *estimate* the likelihood
  - Of a word occurring in a context (N-1)
  - Of a sentence occurring at all
- Although it seems to be simple and rather intuitive, however it is very important and commonly used in many situations such as:
  - Next word guessing
  - ASR (Automatic Speech Recognition)
  - Text Dictation System
  - Text generation
  - QA Chatbot
- Smoothing techniques deal with problems of unseen words in a corpus
- Remember they are all DOMAIN specific, be aware when using that in your application.

Next

Chapter 3 Part-of-Speech Tagging