19CSE453 – Natural Language Processing
Language Model N-grams

By

Ms. Kavitha C.R.

Dept. of Computer Science and Engineering

Amrita School of Computing, Bengaluru

Amrita Vishwa Vidyapeetham

N-grams

N-grams are **continuous sequences of words or symbols or tokens in a document**. In technical terms, they can be defined as the neighboring sequences of items in a document.
They are used when we deal with text data in NLP(Natural Language Processing) tasks.

"Medium blog" is a 2-gram (a bigram),
"A Medium blog post" is a 4-gram, and
"Write on Medium" is a 3-gram (trigram)

Why N-gram though?

Why is it that we need to learn n-gram and the related probability?

Well, in Natural Language Processing, n-grams are used for a variety of applications.

Some examples include

- Auto completion of sentences (such as in mail),
- Auto spell check and
- to a certain extent, can check for grammar in a given sentence.

N-gram Probabilities

Sentence completion system

This system suggests words which could be used next in a given sentence. Suppose given the system a sentence "Thank you so much for your" and expect the system to predict the next word. Now we know that the next word is "help" with a very high probability. But how will the system know that?

One important thing to note here is that, as for any other artificial intelligence or machine learning model, we need to train the model with a huge corpus of data.

Once we do that, the system, or the NLP model will have a pretty good idea of the "probability" of the occurrence of a word after a certain word.

So hoping that we have trained our model with a huge corpus of data, we'll assume that the model gave us the correct answer.

Bigram-model example

Let’s work with a bigram model, and have the following sentences as the training corpus:

1. Thank you so much for your help.
2. I really appreciate your help.
3. Excuse me, do you know what time it is?
4. I’m really sorry for not inviting you.
5. I really like your watch.

After training our model with this data, to write the sentence “I really like your garden.” Now because this is a bigram model, the model will learn the occurrence of every two words, to determine the probability of a word occurring after a certain word.
For example, from the 2nd , 4th and 5th sentence in the example, after the word “really” we see either the word “appreciate”, “sorry”, or “like” occurs. So the model will calculate the probability of each of these sequences.

The probability of word “w2” occurring after the word “w1,” is as follows:

count(w1 w2) / count(w1)

which is the number of times the words occurs in the required sequence, divided by the number of the times the word before the expected word occurs in the corpus.

Bigram-model example

From our example sentences, let's calculate the probability of the word "like" occurring after the word "really":

count(really like) / count(really)

= 1 / 3

= 0.33

Similarly, for the other two possibilities:

count(really appreciate) / count(really)

= 1 / 3

= 0.33

count(really sorry) / count(really)

= 1 / 3

= 0.33

So when we type the phrase "I really," and expect the model to suggest the next word, it'll get the right answer only once out of three times, because the probability of the correct answer is only 1/3.

As an another example, if input sentence to the model is “Thank you for inviting,” and then expect the model to suggest the next word, it’s going to give the word “you” because of the example sentence 4. That’s the only example the model knows.

As you can imagine, if we give the model a **bigger corpus** (or a bigger dataset) to train on, the predictions will improve a lot.

Similarly, we’re only using a bigram here. We can use a trigram or even a 4-gram to improve the model’s understanding of the probabilities.

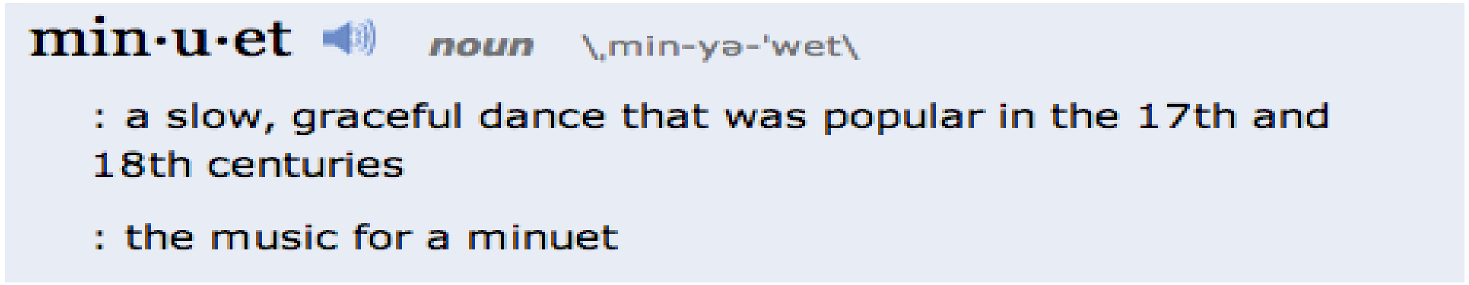
Using these n-grams and the probabilities of the occurrences of certain words in certain sequences could improve the predictions of auto completion systems.

Similarly, NLP and n-grams can be used to train voice-based personal assistant bots.

For example, using a 3-gram or trigram training model, a bot will be able to understand the difference between sentences such as “what’s the temperature?” and “set the temperature”

Context Sensitive Spelling Correction

The office is about fifteen minuets from my house


min·u·et noun \ˌmin-yə-ˈwet\

: a slow, graceful dance that was popular in the 17th and 18th centuries

: the music for a minuet

Use a Language Model

P(about fifteen minutes from) > P(about fifteen minuets from)

Probabilistic Language Models: Applications

Speech Recognition

P(I saw a van) >> P(eyes awe of an)

Machine Translation

Which sentence is more plausible in the target language?

P(high winds) > P(large winds)

Other Applications

- **Context sensitive spelling correction**
- **Natural Language Generation**

Completion Prediction

Language model also supports predicting the completion of a sentence.

- Please turn off your cell ...
- Your program does not ...

Predictive text input systems can guess what you are typing and give choices on how to complete it.

Probabilistic Language Modeling

- **Goal:** Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

- **Related Task:** probability of an upcoming word:

$$P(w_4 | w_1, w_2, w_3)$$

A model that computes either of these is called a language model

Computing P(W)

How to compute the joint probability

P(about, fifteen, minutes, from)

Basic Idea

Rely on the Chain Rule of Probability

The Chain Rule

Conditional Probabilities

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

$$P(A, B) = P(A)P(B|A)$$

Conditional probability is defined as **the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome**.

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

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

Probability of words in sentences

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

P("about fifteen minutes from") =

P(about) * P(fifteen | about) * P(minutes | about fifteen) * P(from | about fifteen minutes)

An N-gram model uses only $N - 1$ words of prior context.

Estimating these Probability Values

Count and divide

$$P(\text{office} \mid \text{about fifteen minutes from}) = \frac{\textit{Count}\ (\text{about fifteen minutes from office})}{\textit{Count}\ (\text{about fifteen minutes from})}$$

What is the problem?

We may never see enough data for estimating these, too many possibilities

Chain-rule applied in joint probability

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

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)

Markov Assumption

Simplifying Assumption: Use only the previous word

P(office | about fifteen minutes from) ≈ P(office | from)

Or the couple previous words

P(office | about fifteen minutes from) ≈ P(office | minutes from)

More Formally: kth order Markov Model

Chain Rule:

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

Using Markov Assumption: only k previous words

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

We approximate each component in the product

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

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
```

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
```

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(s) which I had just put into the machine room on the fifth floor is (are) crashing."

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

Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

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>

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

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

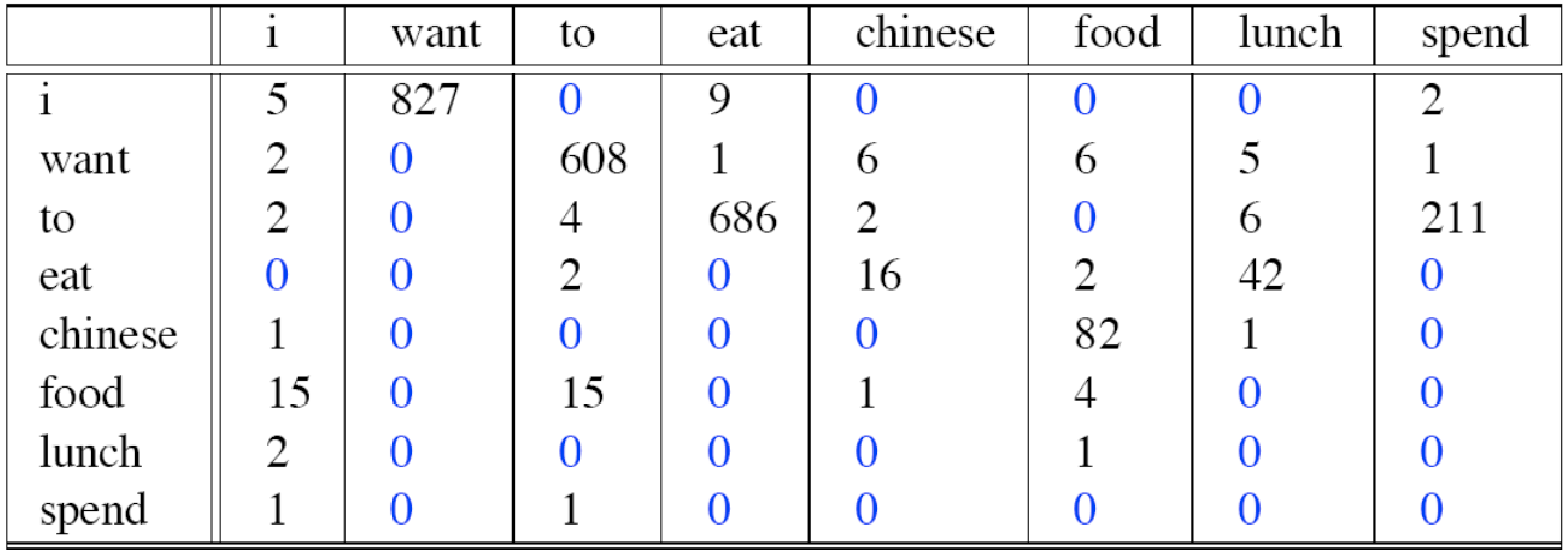
More Examples

- 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

Berkeley Restaurant Project

Raw bigram counts

- Out of **9332** sentences



	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 for eight of the words (out of *V* = 1446) in the Berkeley Restaurant Project corpus of 9332 sentences.

V is **the vocabulary size which is equal to the number of unique words (types) in the corpus**

Raw bigram probabilities

- Normalize by unigrams: (dividing each row by the following unigram counts)

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

Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

Sentence: I want English food or *I want Chinese food*

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

The fact that what comes after *eat* is usually a noun or an adjective, or that what comes after *to* is usually a verb

Practical Issues

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

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

Handling zeros

Use smoothing

Language Modeling Toolkits

- SRILM
  - http://www.speech.sri.com/projects/srilm/
- KenLM
  - https://kheafield.com/code/kenlm/

Example from the 4-gram data

serve as the inspector 66

serve as the inspiration 1390

serve as the installation 136

serve as the institute 187

serve as the institution 279

serve as the institutional 461

Evaluating Language Model

Does it prefer good sentences to bad sentences?

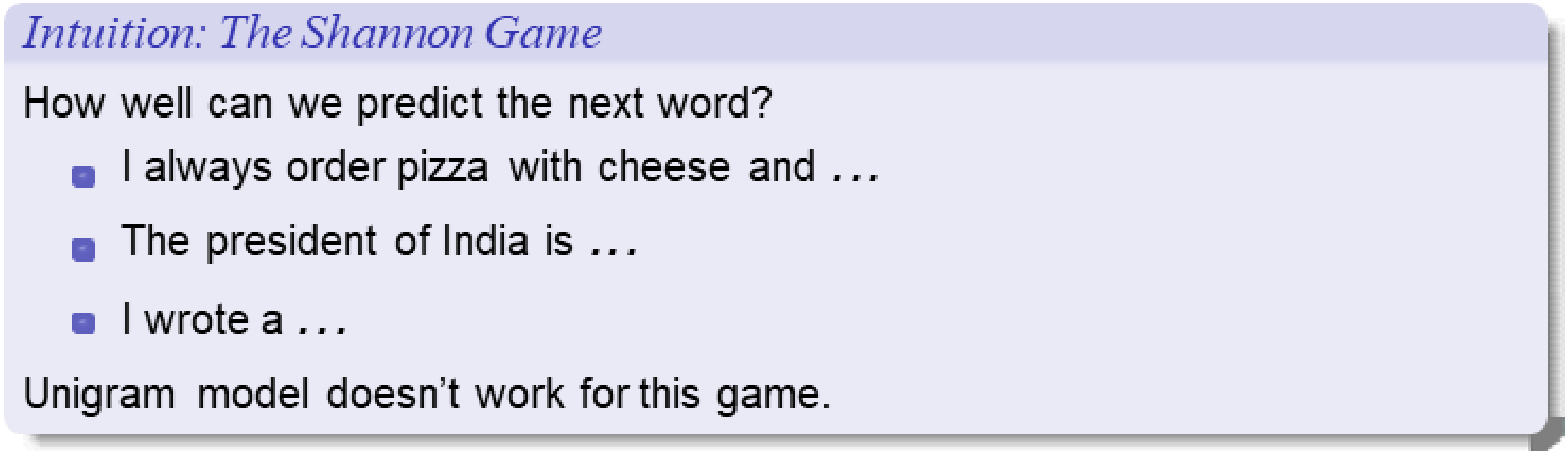
Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Training and Test Corpora

- Parameters of the model are trained on a large corpus of text, called **training set**.
- Performance is tested on a disjoint (held-out) **test data** using an **evaluation metric**

Extrinsic evaluation of N-grams models

The best way to evaluate the performance of a language model is to embed it in an application and measure the total performance of the application.
Such end-to-end evaluation is called **extrinsic evaluation**, and also sometimes called **in vivo** evaluation



Intuition: The Shannon Game

How well can we predict the next word?

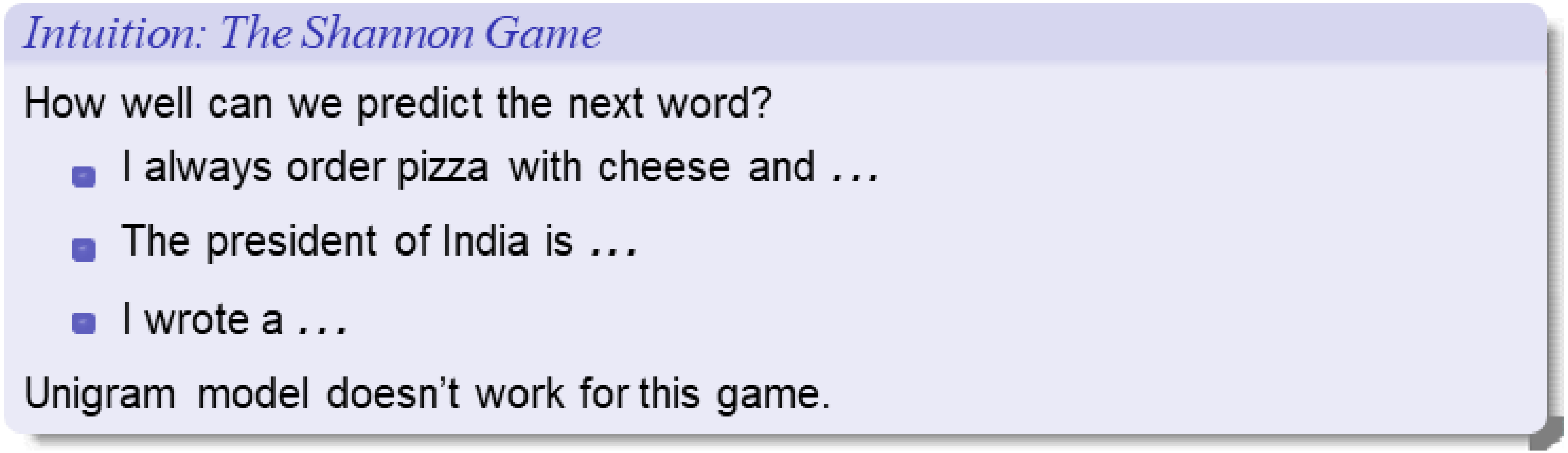
- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Unigram model doesn't work for this game.

Claude E. Shannon. "Prediction and Entropy of Printed English",
***Bell System Technical Journal* 30:50-64. 1951**

Intrinsic evaluation: Perplexity

An intrinsic evaluation metric measures the quality of a model independent of any application

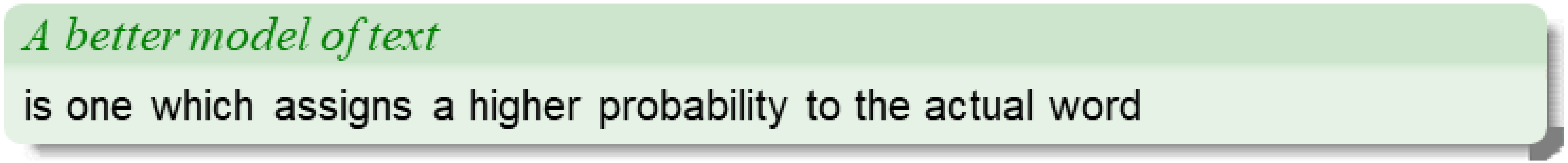


Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and . . .
- The president of India is . . .
- I wrote a . . .

Unigram model doesn't work for this game.



A better model of text

is one which assigns a higher probability to the actual word

Perplexity

Perplexity is a common intrinsic evaluation

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

Perplexity (PP(W))

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

Applying chain Rule

$$PP(W) = \left[\prod \frac{1}{P(w_i|w_1 \ldots w_{i-1})}\right]^{\frac{1}{N}}$$

For bigrams

$$PP(W) = \left[\prod \frac{1}{P(w_i / w_{i-1})}\right]^{1/N}$$

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability p = 1/10 to each digit.

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

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unigram perplexity: 962?

The model is as confused on test data as if it had to choose uniformly and independently among 962 possibilities for each word.

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram (<s>,w) as per its probability
- Choose a random bigram (w,x) as per its probability
- And so on until we choose </s>

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

Problems with simple Maximum Likelihood Estimation: zeros

Training set

- … denied the allegations
- … denied the reports
- … denied the claims
- … denied the request

Test Data

- … denied the offer
- … denied the loan

Zero probability n-grams

- P(offer | denied the) = 0
- The test set will be assigned a probability 0
- And the perplexity can't be computed

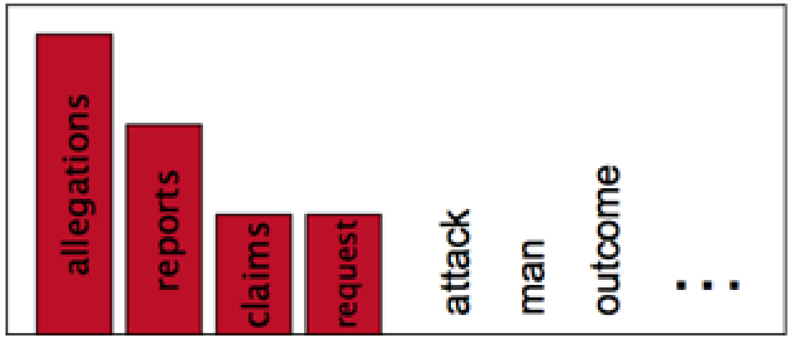
Language Modeling: Smoothing

- **Develop a model which decreases probability of seen events and allows the occurrence of previously unseen n-grams**
- **Also known as "Discounting methods"**
- **"Validation" – Smoothing methods which utilize a second batch of test data.**

Language Modeling: Smoothing

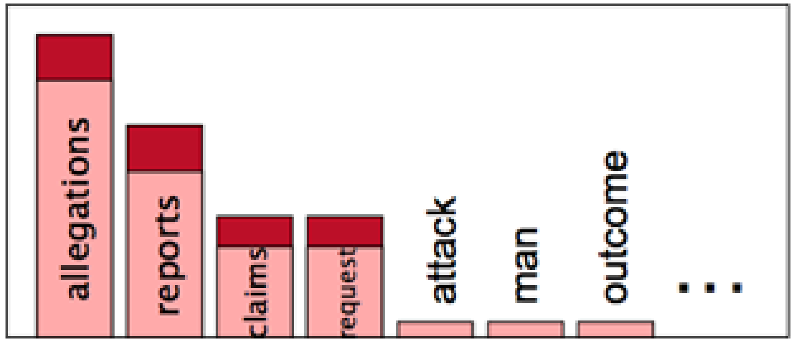
With sparse statistics

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



Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time that we actually did
- Just add one to all the counts!
- MLE estimate for bigram: $P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$ Count(W[i-1]W[i])/Count(W[i-1])
- Add-1 estimate: $P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)+1}{c(w_{i-1})+V}$ (Count(W[i-1]W[i])+1)/(Count(W[i-1])+V)

V is **the vocabulary size which is equal to the number of unique words (types) in your corpus**

MLE-Maximum Likelihood Estimation

Reconstituted counts as effect of smoothing

Effective bigram count $(c^*(w_{n-1}w_n))$

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

Comparing with bigrams: Restaurant corpus

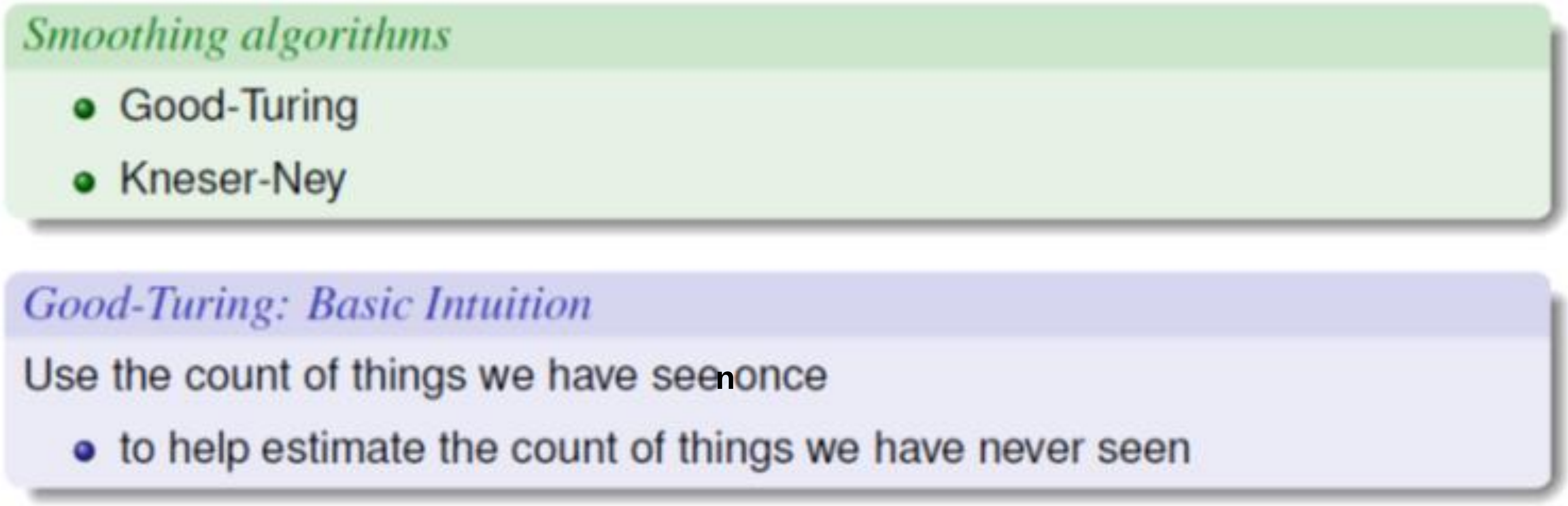
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	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

More general formulations: Add-k

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

Advanced smoothing algorithms



Smoothing algorithms

- Good-Turing
- Kneser-Ney

Good-Turing: Basic Intuition

Use the count of things we have seen once

- to help estimate the count of things we have never seen

N_c: Frequency of frequency c

Example Sentences

<s>I am here </s>

<s>who am I </s>

<s>I would like </s>

Computing N_c

I	3
am	2
here	1
who	1
would	1
like	1

$N_1 = 4$
$N_2 = 1$
$N_3 = 1$

Good Turing Estimation

Idea

Reallocate the probability mass of n−grams that occur $r+1$ times in the training data to the n−grams that occur r times

In particular, reallocate the probability mass of n−grams that were seen once to the n−grams that were never seen

Adjusted count

For each count c, an adjusted count c^* is computed as:

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

where N_c is the number of n−grams seen exactly c times

Good Turing Estimation

Good Turing Smoothing

P^*_{GT}(things with frequency c) = $\frac{c^*}{N}$

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

Exercise

Given the Data set.

He is Sleeping
He is Running
He is Jumping
He is Reading
He is not writing

1. Write the unigrams, bigrams for the given data set
2. Estimate the probability using the unigram, bigram model
3. Predict the next word for the sentence “He is not” using bigram model
4. Check whether smoothing is required or not for the bigram model. If so, apply Laplace Smoothing and Good Turing smoothing and compare.

Thank You