

# N-gram models for text prediction

# Probabilistic Language Models

Today's goal: assign a probability to a sentence

Applications:

- Machine Translation:
  - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
- Spell Correction
  - The office is about fifteen minuets from my house
    - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- Speech Recognition
  - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
- + Summarization, question-answering, etc., etc.!!

# Main assumption

- Sentences follow regular sequences of words
- Such a regularity increases within a given domain (e.g., technical language, chat messages, etc.)
- Even more, the regularity may increase within the same user
- That's why your smartphone keyboard is able to “learn” from your text

# Even for source code...

## On the Naturalness of Software

Abram Hindle, Earl T. Barr, Zhendong Su  
*Dept. of Computer Science*  
*University of California at Davis*  
*Davis, CA 95616 USA*  
*{ajhindle,barr,su}@cs.ucdavis.edu*

Mark Gabel  
*Dept. of Computer Science*  
*The University of Texas at Dallas*  
*Richardson, TX 75080 USA*  
*mark.gabel@utdallas.edu*

Premkumar Devanbu  
*Dept. of Computer Science*  
*University of California at Davis*  
*Davis, CA 95616 USA*  
*devanbu@cs.ucdavis.edu*

**Abstract**—Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.

We begin with the conjecture that most software is also natural, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers. Using the widely adopted n-gram model, we provide empirical

efforts in the 1960s. In the '70s and '80s, the field was re-animated with ideas from logic and formal semantics, which still proved too cumbersome to perform practical tasks at scale. Both these approaches essentially dealt with NLP from first principles—addressing *language*, in all its rich theoretical glory, rather than examining corpora of actual *utterances*, *i.e.*, what people actually write or say. In the 1980s, a fundamental shift to *corpus-based*, *statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including “aligned” text with translations in multiple languages,<sup>1</sup> along with the computational muscle (CPU speed, primary and secondary storage) to estimate robust statistical models over very large data sets has led to stunning progress and widely-available practical applications, such as statistical translation used by [translate.google.com](http://translate.google.com).<sup>2</sup> We argue that an essential fact underlying this modern, exciting phase of

# Probabilistic Language Modeling

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

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

A model that computes either of these:

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

is called a **language model**.

# How to compute $P(W)$

How to compute this joint probability:

**$P(\text{its, water, is, so, transparent, that})$**

Intuition: let's rely on the Chain Rule of Probability

# Reminder: The Chain Rule

Recall the definition of conditional probabilities

$$p(\mathbf{B}|\mathbf{A}) = \mathbf{P}(\mathbf{A},\mathbf{B})/\mathbf{P}(\mathbf{A})$$

Rewriting:  $\mathbf{P}(\mathbf{A},\mathbf{B}) = \mathbf{P}(\mathbf{A})\mathbf{P}(\mathbf{B}|\mathbf{A})$

More variables:

$$P(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}) = P(\mathbf{A})P(\mathbf{B}|\mathbf{A})P(\mathbf{C}|\mathbf{A},\mathbf{B})P(\mathbf{D}|\mathbf{A},\mathbf{B},\mathbf{C})$$

The Chain Rule in General

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

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

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

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water})$

$\times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so})$



# How to estimate these probabilities

Could we just count and divide?

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

No! Too many possible sentences!

We'll never see enough data for estimating these

# Markov Assumption



Andrei Markov

Simplifying assumption:

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

Or maybe

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

# 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

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

# Simplest case: Unigram model

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

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:

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

texaco, rose, one, in, this, issue, is, pursuing,  
growth, in, a, boiler, house, said, mr., gurrria,  
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 which I had just put into the machine room on the fifth floor crashed.”

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

# Language Modeling

Estimating N-gram Probabilities

# Estimating bigram probabilities

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

The Maximum Likelihood Estimate

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



# An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$$\begin{array}{lll} P(\text{I} | \text{<s>}) = \frac{2}{3} = .67 & P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33 & P(\text{am} | \text{I}) = \frac{2}{3} = .67 \\ P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5 & P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 & P(\text{do} | \text{I}) = \frac{1}{3} = .33 \end{array}$$

# Practical Issues

We do all computations in log space

- Avoid underflow
- (also adding is faster than multiplying)

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

# 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.

# Language Modeling

Evaluation and Perplexity

# Evaluation:

## How good is our model?

- 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
- Therefore:
  - 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

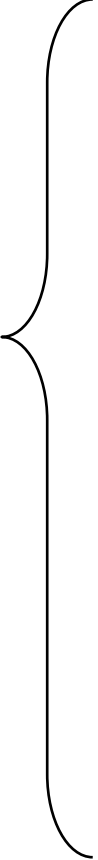
## The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_

I saw a \_\_\_\_



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

Unigrams are terrible at this game. (Why?)

A better model of a text

- is one which assigns a higher probability to the word that actually occurs



# Perplexity

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

Chain rule:

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

For bigrams:

$$\begin{aligned} PP(W) &= \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \\ PP(W) &= \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}} \end{aligned}$$

**Minimizing perplexity is the same as maximizing probability**

# 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)!

# Language Modeling

Smoothing: Add-one (Laplace) smoothing

# The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

$P(w \mid \text{denied the})$

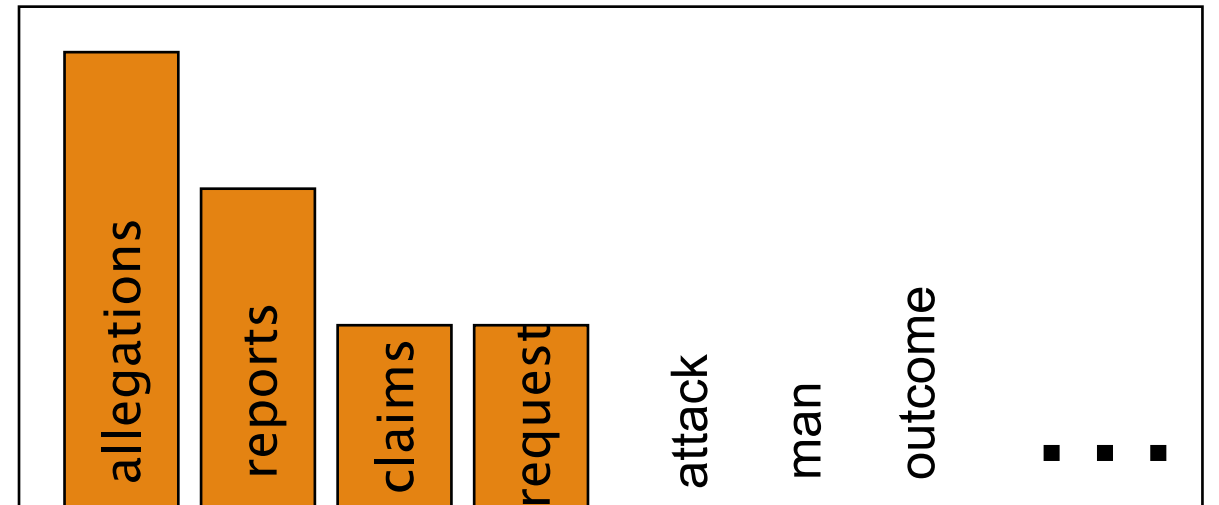
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

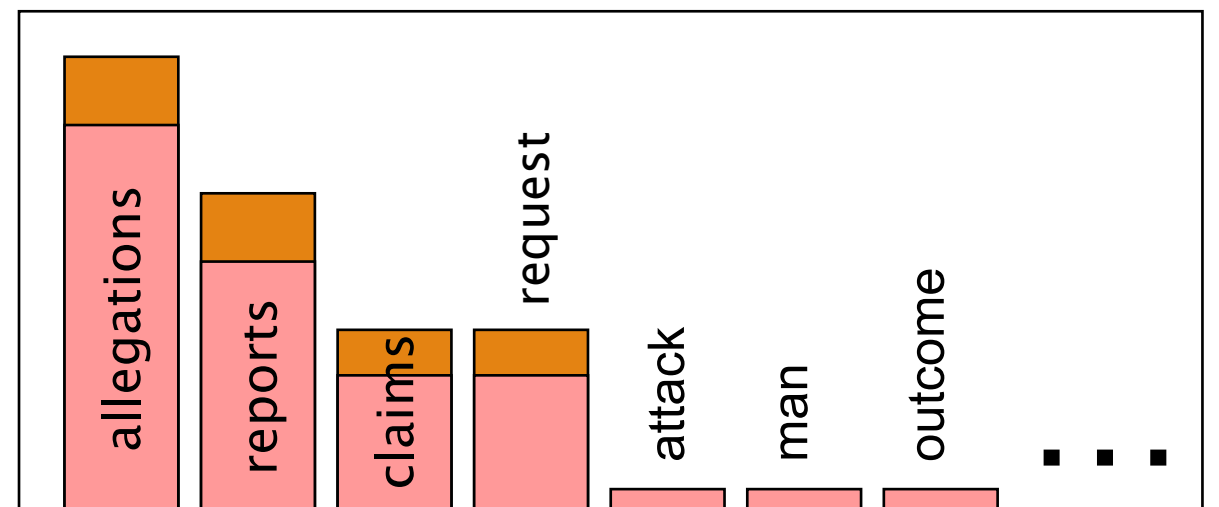
1.5 reports

0.5 claims

0.5 request

2 other

7 total



# Practical applications

The nltk lm library

# The nltk lm library

- To apply n-gram language models in practice, we will rely on the nltk lm library
- More in the attached notebook

# Getting bigrams

```
text = [['I', 'want', 'to', 'go', 'home'], ['This', 'file',  
'contains', 'a', 'critical', 'bug']]
```

Getting all bigrams

```
from nltk.util import bigrams  
list(bigrams(text[0]))
```

```
[('I', 'want'), ('want', 'to'), ('to', 'go'), ('go', 'home')]
```

# Padding sentences

Useful to create a sentence begin and end token

```
from nltk.lm.preprocessing import pad_both_ends
paddedSent=list(pad_both_ends(text[0], n=2))
print(paddedSent)
['<s>', 'I', 'want', 'to', 'go', 'home', '</s>']
```

Bigrams again...

```
list(bigrams(paddedSent))
[('<s>', 'I'),
 ('I', 'want'),
 ('want', 'to'),
 ('to', 'go'),
 ('go', 'home'),
 ('home', '</s>')]
```



# Everygrams

```
from nltk import everygrams
list(everygrams(paddedSent, max_len=3))
[('<s>',),
 ('<s>', 'I'),
 ('<s>', 'I', 'want'),
 ('I',),
 ('I', 'want'),
 ('I', 'want', 'to'),
 ('want',),
 ('want', 'to'),
 ('want', 'to', 'go'),
 ('to',),
 ('to', 'go'),
 ('to', 'go', 'home'),
 ('go',),
 ('go', 'home'),
 ('go', 'home', '</s>'),
 ('home',),
 ('home', '</s>'),
 ('</s>',)]
```

**Using a  
non-trivial corpus**

# Loading the dataset

Let's use Trump Tweets  
from here <https://github.com/MarkHershey/CompleteTrumpTweetsArchive>

```
with open("tweets.csv") as f:  
    lines=f.readlines()
```

# Cleanup function

```
import re
import tqdm
from nltk import sent_tokenize, word_tokenize
def cleanUp(text):
    #remove newlines
    text=text.strip()
    #remove tags
    text=re.sub("[@\\#]\\S+", "", text)
    #remove URLs
    text=re.sub("https?:\\/\\/\\S+", "", text)
    text=re.sub("pic\\.twitter\\.com\\S+", "", text)
    #tokenize the tweet into sentences
    sentences=sent_tokenize(text)
    corpus=[]
    for sentence in sentences:
        words=word_tokenize(sentence)
        #take only words from tweets
        cleaned=[w.lower() for w in words if re.search("\\w+", w)]
        corpus.append(cleaned)
    return corpus
```

# Cleaning all sentences

```
allSentences=[]  
for line in lines:  
    allSentences.extend(cleanUp(line))
```

# Training set and model

- Creating the training set and the vocabulary, up to 3-grams:

```
from nltk.lm.preprocessing import padded_everygram_pipeline
train, vocab = padded_everygram_pipeline(3, allSentences)
```

- Model fitting:

```
from nltk.lm import MLE
TrumpModel = MLE(3)
TrumpModel.fit(train, vocab)
```

# Inference examples...

```
TrumpModel.generate(2, ["i", "will"], random_seed=3)
```

```
['be', 'interviewed']
```

```
TrumpModel.generate(2, ["make", "america"], random_seed=5)
```

```
['great', 'again']
```

```
TrumpModel.perplexity([['make', 'america'], ['great', 'again']])
```

```
6.7922304393542845
```

```
TrumpModel.perplexity([['make', 'america'], ['healthy']])
```

```
382.68996985395387
```

```
TrumpModel.counts[['great']]['again']
```

```
273
```