Words, Regular Expressions, and N-gram (Markov) Models

CS6120: Natural Language Processing
Northeastern University

Si Wu

(borrowed some slides created by David Smith, and slides from Jurafsky & Martin Chapter 2)

Logistics

- Class website https://siwu.io/nlp-class/
- Make sure you join
  - Gradescope, for coding assignment, code to join **NGZDZP**
  - Ed Discussion, for asking questions, link on class website
- We are *not* using Canvas
- **Office hours** are posted on the class website
  - Any preference on in-person vs. remote?
- Attend lectures, there will be in-class quizzes

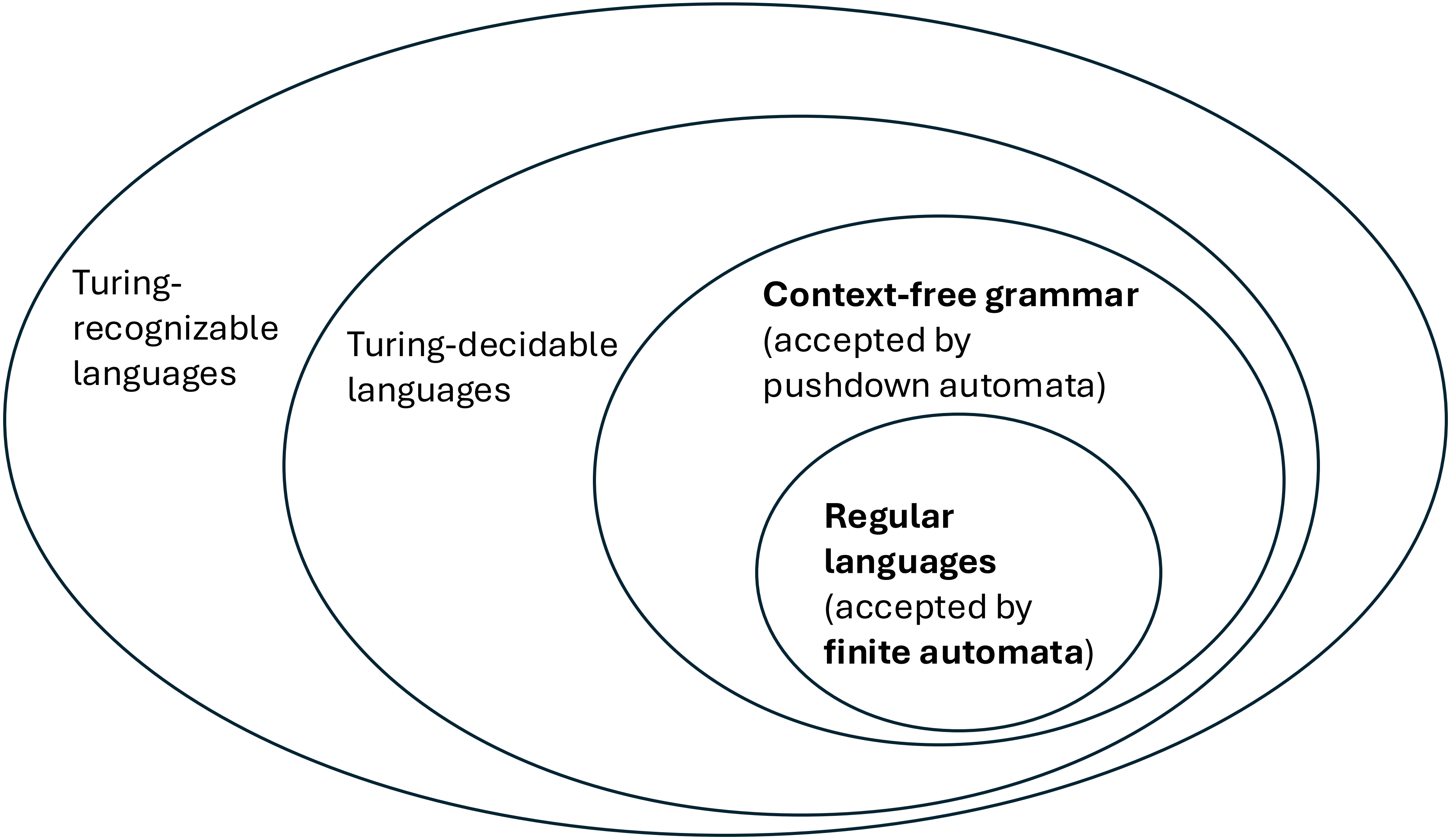
Review

- Last time we briefly talked about
    - Why human language is difficult to study
    - The social nature of human languages
    - The statistical nature of language models
- We also briefly mentioned **entropy**, naïve bayes, neural network, and LLMs.
- In the next 3 lectures, we will talk about strings, n-grams, and some classic and “simpler” statistical language models

Regular expression

Talking about strings

Chomsky Hierarchy in Theory of Computation



Turing-recognizable languages

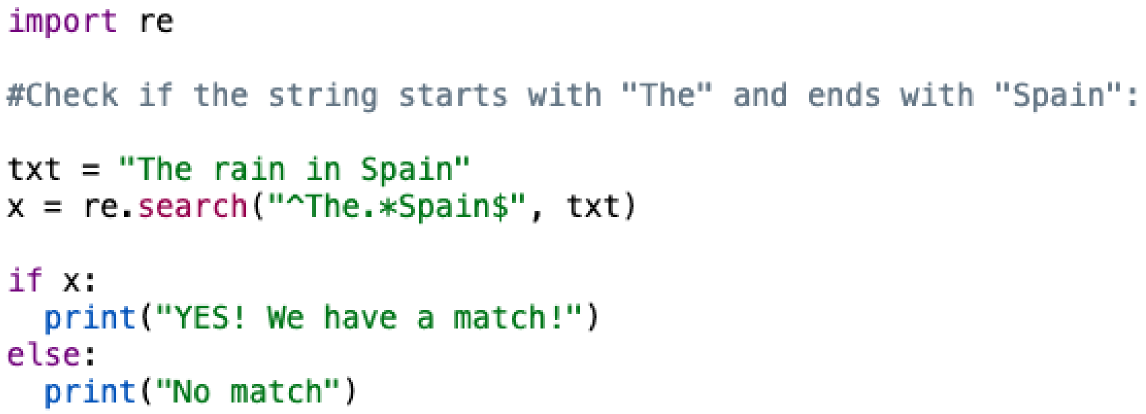
Turing-decidable languages

Context-free grammar (accepted by pushdown automata)

Regular languages (accepted by **finite automata**)

Regular expressions are used everywhere

- A formal language for specifying text strings
- Part of every text processing task
  - Often a useful pre-processing or text formatting step, for example for BPE tokenization
- Also necessary for data analysis of text
- A widely used tool in industry and academics



```
import re

#Check if the string starts with "The" and ends with "Spain":

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

if x:
  print("YES! We have a match!")
else:
  print("No match")
```

Theory of Computation Refresh

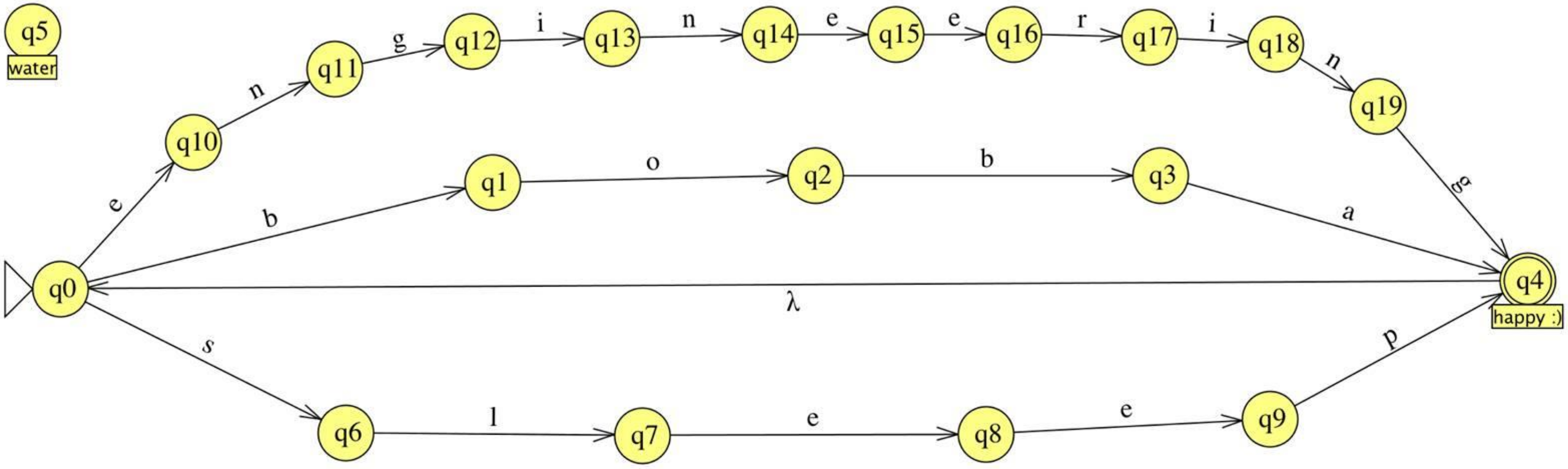
- Any **regular expression** can be expressed in a **finite-state automaton** (deterministic or non-deterministic), and vice versa
- Def: **Finite state automaton/machine**:
  - Input: finite string over a fixed set of acceptable symbols
  - Output: accept or reject
  - Memory is limited by the num of states it has
- **Regular language**: the set of accepted strings defined by the regular expression.

Theory of Computation refresh

- Kleene star: (ba)* → “”, “ba”, “baba”, etc.
- Kleene plus: at least once. (ba)+ → “ba”, “baba”, etc.
- Regular language is closed under
  - Negation/complement
  - Addition/concatenation
  - Union
  - Intersection
  - Reversal
  - Etc.

A funny but intuitive example of finite state automation

(made with JFLAP, circa 2018)



Accepted strings: “engineering”, “boba”, “sleep”

A **language model** is a function that assigns a probability to a string of text.

S = { The quick brown fox jumped over the lazy dog }

$P(s) = 1$ if $s \in S$

$P(s) = 0$ otherwise

S = {

The quick brown fox…,

When in the course of human events…,

It was a bright cold day in April and the clocks…

}

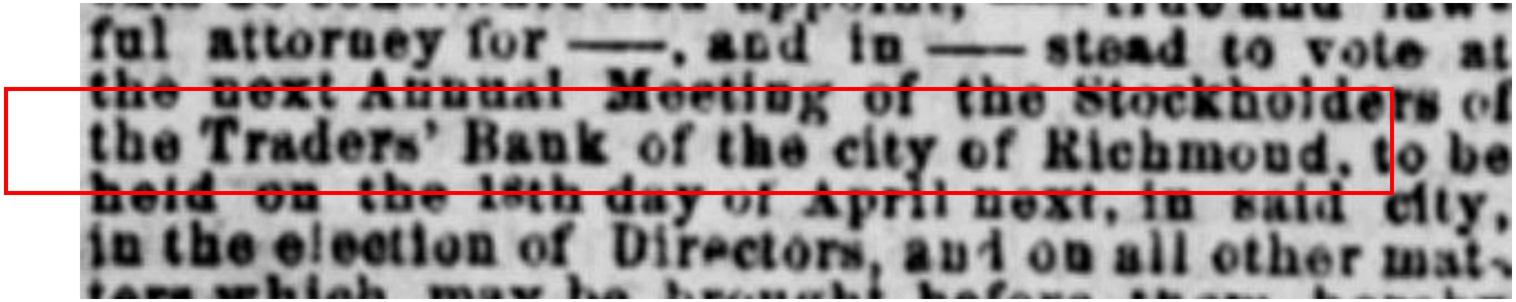
$$P(s) = \frac{1}{|S|} \text{ if } s \in S$$

$$P(s) = 0 \text{ otherwise}$$

This works for finite sets

Strings as Queries

You’re looking at old financial notices:

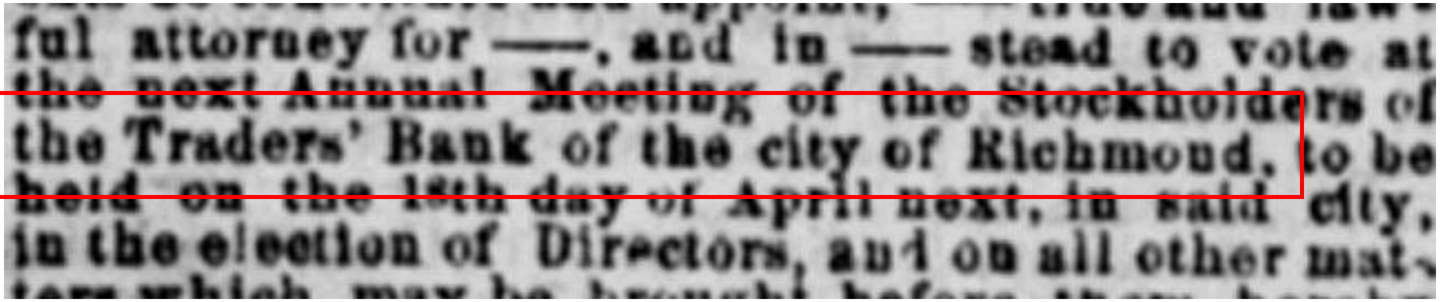


ful attorney for ——, and in —— stead to vote at
the next Annual Meeting of the Stockholders of
the Traders' Bank of the city of Richmond, to be
held on the 18th day of April next, in said city,
in the e!ection of Directors, an1 on all other mat-

searching for:

```
the Traders’ Bank of the city of Richmond
```

Strings as Queries



ful attorney for ——, and in —— stead to vote at
the next Annual Meeting of the Stockholders of
the Traders' Bank of the city of Richmond, to be
held on the 18th day of April next, in said city,
in the election of Directors, and on all other mat-

But these lines get transcribed as:

```
the Trader. Bank of the city of Richmoud, to be
tbe Traders' Bank or the city of Biebmond, to bo
tbe Traders' Bank of the city of Klchmoud, to be,
the Traders' Hank of the city of Richmoud, lo be j
the Trader*' Bsnk of the city of Richmond, to be
the Traders' Hank of the city of Richmond, to he
tha Traders' Bank of the cltv of Richmond to be
```

Exact match won't work!

Generalized Queries

Notice confusion of c/e/o, b/h, B/H/K/R:

```
the Trader. Bank of the city of Richmoud, to be
tbe Traders' Bank or the city of Biebmond, to bo
tbe Traders' Bank of the city of Klchmoud, to be,
the Traders' Hank of the city of Richmoud, lo be j
the Trader*' Bsnk of the city of Richmond, to be
the Traders' Hank of the city of Richmond, to he
tha Traders' Bank of the cltv of Richmond to be
```

Instead of searching for:

```
the Traders’ Bank of the city of Richmond
```

Search this:

```
t[bh][ceo] Trad[ceo]rs’ [BHKR]ank o[fr] th[ceo] [ceo]ity
[ceo][fr] [BHKR]i[ceo][bh]m[ceo]nd
```

Generalized Queries

Using this expression,

```
t[bh][ceo] Trad[ceo]rs’ [BHKR]ank o[fr]
th[ceo] [ceo]ity [ceo][fr]
[BHKR]i[ceo][bh]m[ceo]nd
```

We will match two of them:

the Trader. Bank of the city of Richmoud, to be
tbe Traders' Bank or the city of Biebmond, to bo
tbe Traders' Bank of the city of Klchmoud, to be,
the Traders' Hank of the city of Richmoud, lo be j
the Trader*' Bsnk of the city of Richmond, to be
the Traders' Hank of the city of Richmond, to he
tha Traders' Bank of the cltv of Richmond to be

Regular Languages

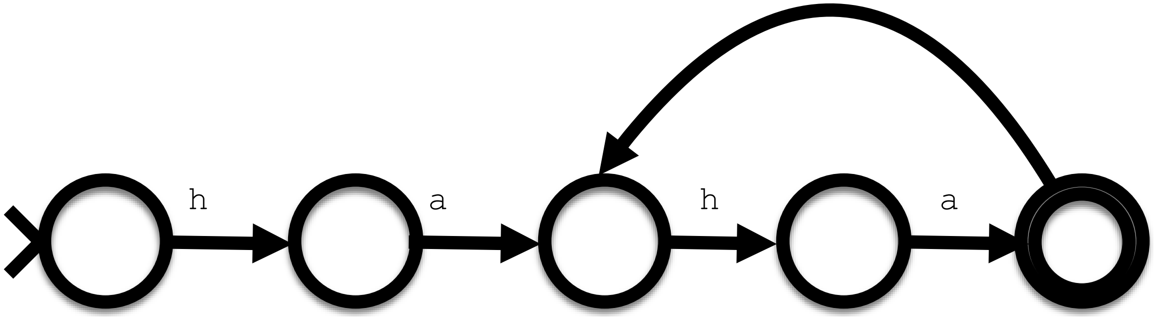
S = {
    haha,
    hahaha,
    hahahaha,
    hahahahaha,
    ...
}

Kleene plus

`ha(ha)+`

syntactic sugar for
Kleene star

`haha(ha)*`



Regular Languages

But this regular language

```
t[bh][ceo] Trad[ceo]rs’ [BHKR]ank o[fr]
th[ceo] [ceo]ity [ceo][fr]
[BHKR]i[ceo][bh]m[ceo]nd
```

There are

2*3*3*4*2*3*3*3*2*4*3*2*3=559,872

Possible variations

Surely some strings are more likely!

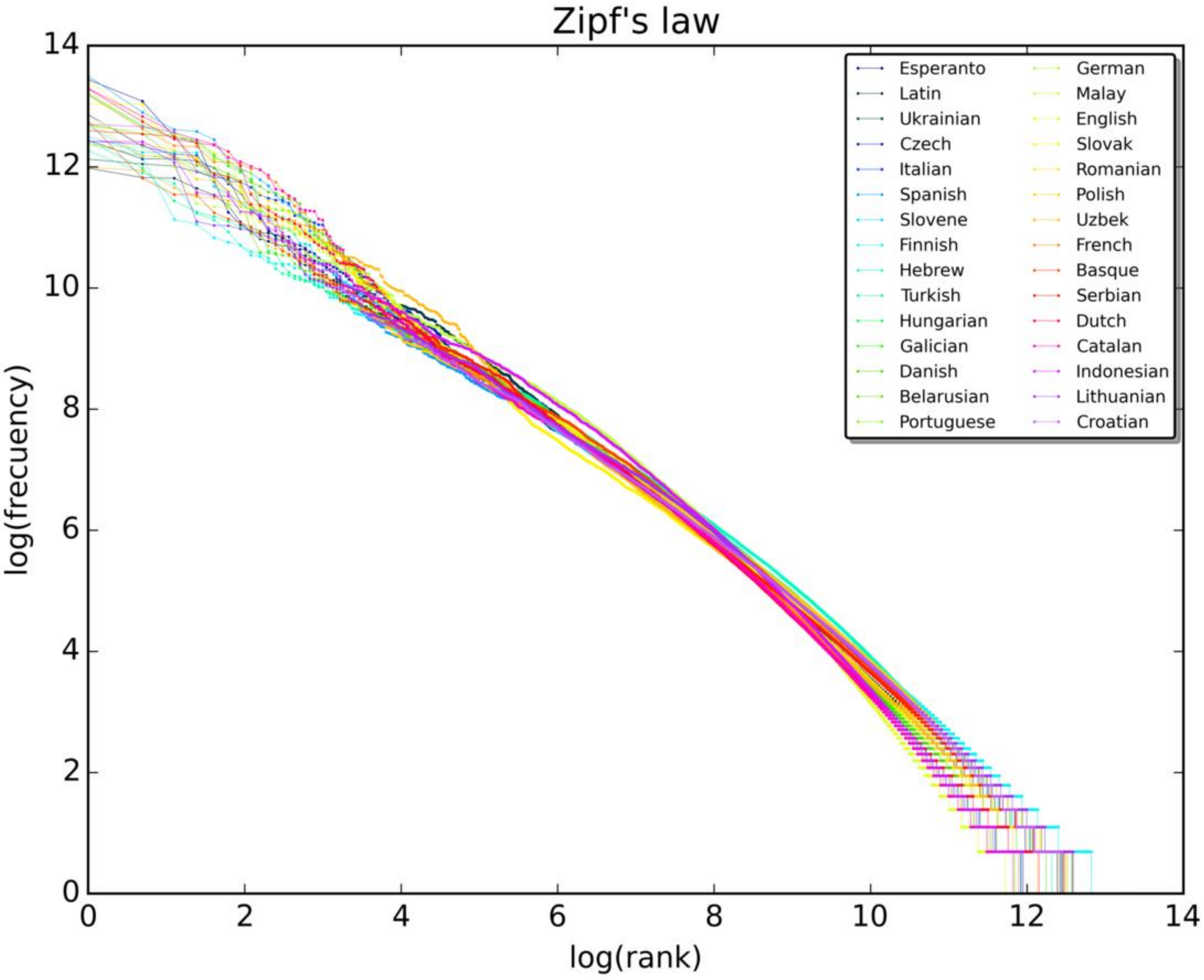
Zipf’s Law

$$\text{word frequency} \propto \frac{\text{constant}}{\text{word rank}}$$

- **Distribution of word frequencies is very *skewed***
  - a few words occur very often, many words hardly ever occur
  - e.g., two most common words (“the”, “of”) make up about 10% of all word occurrences in text documents

Zipf’s law (more generally, a “power law”):

- Given that words are ranked in order of decreasing frequency
- **Observe that rank (*r*) of a word times its frequency (*f*) is approximately a constant (*k*)**
- i.e., $r \cdot f \approx k$ **or** $r \cdot P_r \approx c$**, where** P_r **is relative frequency of word occurrence and** $c \approx 0.1$ **for English**



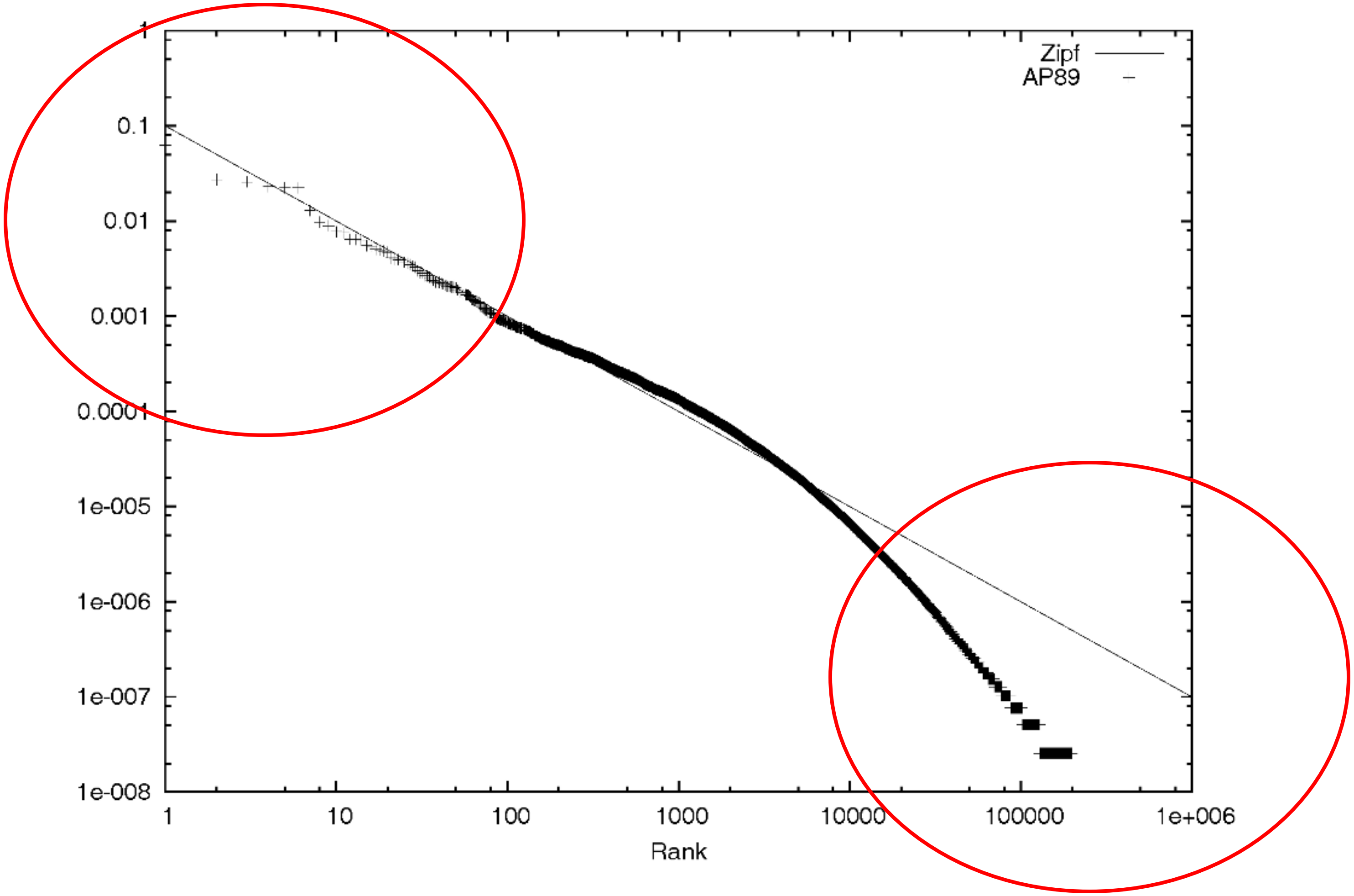
From Wikipedia, Zipf's law plot for the first 10 million words in 30 Wikipedias (as of October 2015) in a log-log scale

AP89 (Associated Press 1989) Example

Total documents	84,678
Total word occurrences	39,749,179
Vocabulary size	198,763
Words occurring > 1000 times	4,169
Words occurring once	70,064

Word	*Freq.*	*r*	*Pr(%)*	*r.Pr*
assistant	5,095	1,021	.013	0.13
sewers	100	17,110	2.56 × 10–4	0.04
toothbrush	10	51,555	2.56 × 10–5	0.01
hazmat	1	166,945	2.56 × 10–6	0.04

Zipf’s Law for AP89



log–log plot: note deviations at high and low frequencies

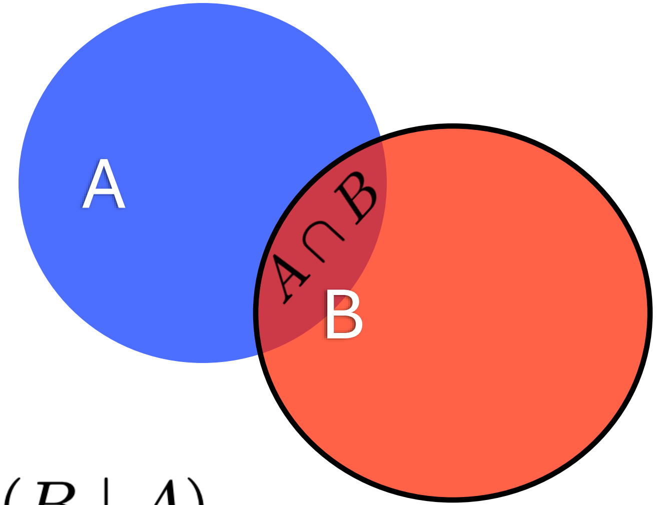
Probability

Axioms of Probability

- Define event space $\bigcup_i \mathcal{F}_i = \Omega$
- Probability function, s.t. $P : \mathcal{F} \rightarrow [0, 1]$
  - Sum of disjoint events A, B $A \cap B = \emptyset \Leftrightarrow P(A \cup B) = P(A) + P(B)$
  - All events sum to one $P(\Omega) = 1$
- Show that: $P(\bar{A}) = 1 - P(A)$

Conditional Probability

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$



$$P(A \cap B) = P(B)P(A \mid B) = P(A)P(B \mid A)$$

Chain rule

$$P(A_1 \cap A_2 \cap \cdots \cap A_n) = P(\underline{A_1})\,P(\underline{A_2} \mid A_1)\,P(\underline{A_3} \mid A_1 \cap A_2) \cdots P(\underline{A_n} \mid A_1 \cap A_2 \cap \cdots \cap A_{n-1})$$

Independence

$$
\begin{array}{ccc}
P(A,B) & = & P(A)P(B) \\
& \Leftrightarrow & \\
P(A \mid B) = P(A) & \wedge & P(B \mid A) = P(B)
\end{array}
$$

In coding terms, knowing *B* doesn’t help in decoding *A*, and vice versa.

A: tomorrow is sunny , P(A) = 0.5
B: the US president will post on Twitter, P(B) = 0.5
P(A|B) = P(A) because the events A and B are independent

Markov Models

$$p(w_1, w_2, \dots, w_n) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)$$
$$\cdot\, p(w_4|w_1, w_2, w_3) \cdots p(w_n|w_1, \dots, w_{n-1})$$

First-order Markov independence assumption

$$p(w_i|w_1, \dots, w_{i-1}) \approx p(w_i \mid w_{i-1})$$

Bigram!

$$p(w_1, w_2, \dots, w_n) \approx p(w_1)p(w_2|w_1)p(w_3|w_2)$$
$$\cdot\, p(w_4 \mid w_3) \cdots p(w_n \mid w_{n-1})$$

First-order Markov assumption → bigram model , only depends on the word before

$$p(w_i \mid w_1, \ldots, w_{i-1}) \approx p(w_i \mid w_{i-1})$$

Second-order Markov assumption → trigram model , two words before

$$p(w_i \mid w_1, \ldots, w_{i-1}) \approx p(w_i \mid w_{i-2}, w_{i-1})$$

- “The train is late again”

- P(again | the, train, is, late) =
  - In bi-gram, or first-order Markov assumption

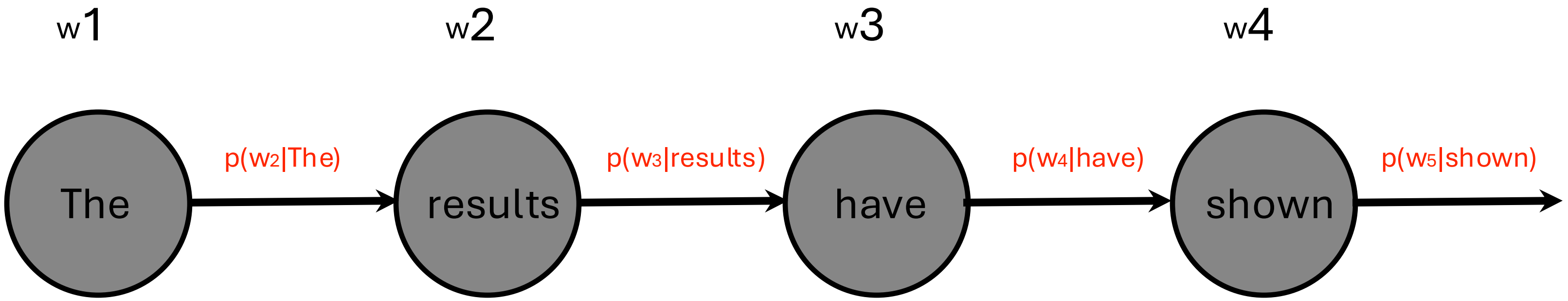
    p(again | the, train, is, late) = p(again | late)

  - In tri-gram, or second-order Markov assumption

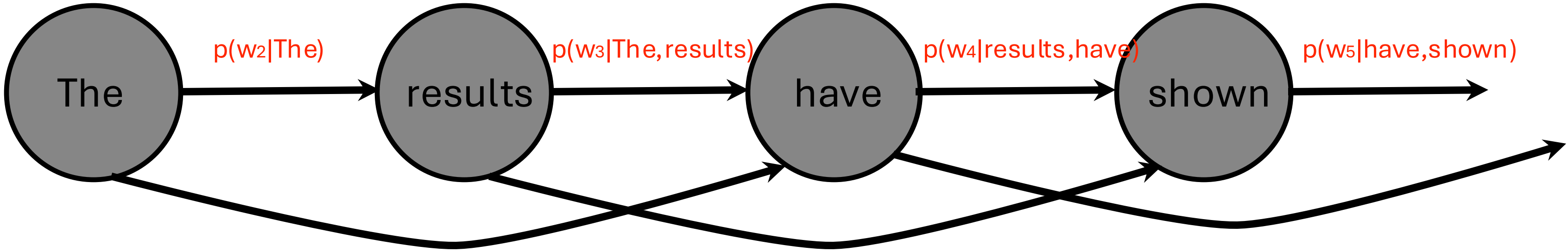
    p(again | the, train, is, late) = p(again | is, late)

Another View

Directed graphical models: *lack* of edge means conditional independence

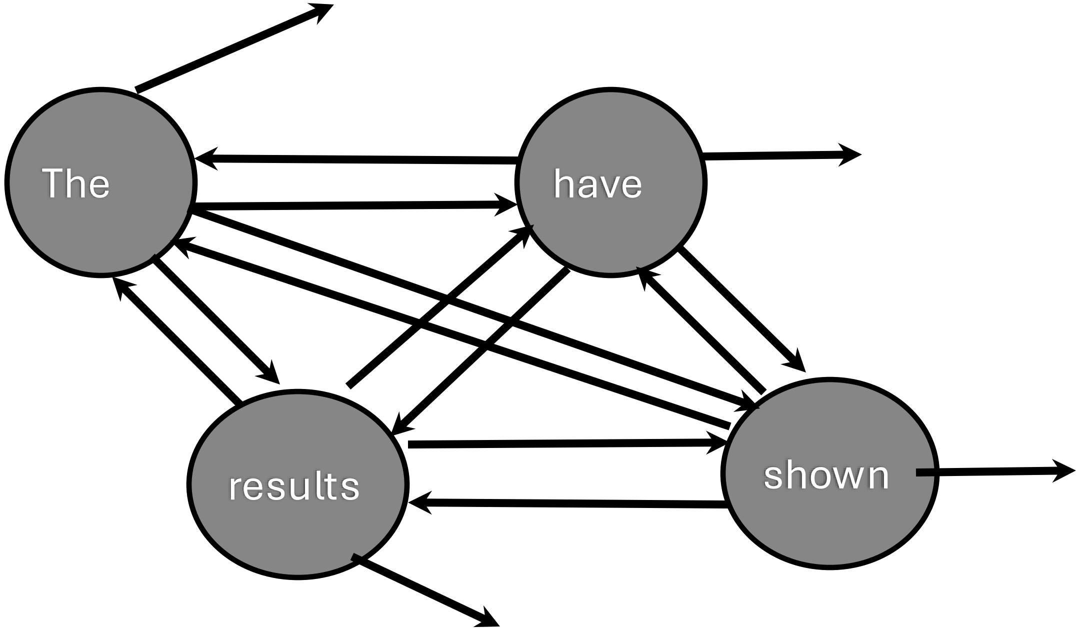


Bigram model as (dynamic) Bayes net



Trigram model as (dynamic) Bayes net

Yet Another View



Bigram model as finite state machine

What about a trigram model?

Bayesian classifier

And examples of sentiment analysis

Movie Reviews

+ Dune Part 1 transcends the boundaries of modern cinema, delivering an **awe-inspiring** journey into the heart of a **mesmerizing** universe. Director Denis Villeneuve's visionary adaptation of Frank Herbert's **iconic** novel captivates from the opening scene to the **breathtaking** finale. With **stunning** visuals, **meticulous** attention to detail, and a cast that brings each character to life with **unparalleled** depth…

— Personally, I've never read the book by Frank Herbert, yet a movie based on it's original source material has to work on it's own. I am not one of those people who adore "Dune," It is such a **tedious** dud that left me **bored** to **death**. It was so **boring**, that it would make it's original novel a **weary** bedtime story. Don't get me wrong, It's not really a **bad** movie by any means (it does have some adequate moments, there are some redeemable qualities, and there is even some inventive creativity), but…

— Yet another **watered** version of the already **watered** down late Star Wars saga, now even more **shamefully** targeted to the teen public, coated with great scenes, great performances and top celebs. The protagonist depicts once again the typical **frail** and introspective teenager whose deeper egoistic dream is to be the center of the universe, full of power, glory and in a reality where adults are basically **stupid**, **nuisances** or **stereotypical buffoons**. For me this type of story just keeps cashing in over…

Setting up a Classifier

What we want:

$p(+ \mid w_1, w_2, \ldots, w_n) > p(- \mid w_1, w_2, \ldots, w_n)$?

What we know how to build:

- A language model for each class
  - $p(w_1, w_2, \ldots, w_n \mid +)$
  - $p(w_1, w_2, \ldots, w_n \mid -)$

Likelihood

Bayes' Theorem

By the definition of conditional probability:

$$P(A, B) = P(B)P(A \mid B) = P(A)P(B \mid A)$$

we can show:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Rev. T. Bayes

A “Bayesian” Classifier

Nowadays also means modeling uncertainty about *p*

$$p(R \mid w_1, w_2, \ldots, w_n) = \frac{p(R)p(w_1, w_2, \ldots, w_n \mid R)}{p(w_1, w_2, \ldots, w_n)}$$

$$\max_{R\in\{\ddot{\smile},\ddot{\frown}\}} p(R \mid w_1, w_2, \ldots, w_n) = \max_{R\in\{\ddot{\smile},\ddot{\frown}\}} p(R)p(w_1, w_2, \ldots, w_n \mid R)$$

Posterior

Prior

Likelihood

P (comedy | “this is a great movie”) =
P(comedy) P(“this is a great movie” | comedy)

- Posterior: during prediction, “given this text, what’s the probability it belongs to this class”.
- Prior: the probability of this genre, how common is this genre
- Likelihood: What language models learn. It learns what the text that belongs to this class looks like. This is the “fit score” of a data/review to a class.

Naive Bayes Intuition

- Simple ("naive") classification method based on Bayes rule
- Relies on very simple representation of document
  - **Bag of words**

- Different types of naïve Bayes:
  - Counting word occurrence, e.g. "so so great", so 2, great 1→ **multinomial naïve bayes.** In this class, we say counting **tokens**
  - Not counting word occurrence, e.g. "so so great", so 1, great 1 → **Bernoulli naïve bayes.** In this class, we say counting word **types**

The bag of words representation

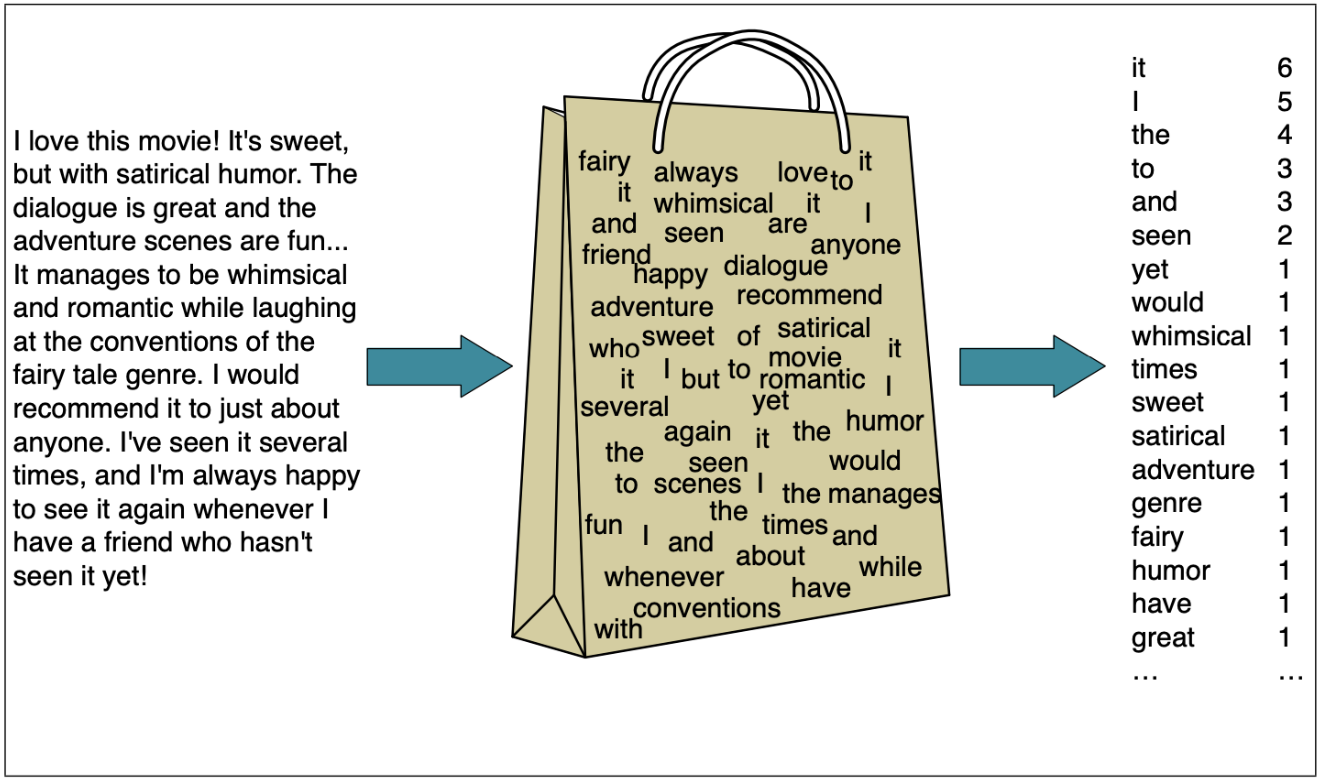
$\gamma($

seen	2
sweet	1
whimsical	1
recommend	1
happy	1
...	...

$)=c$

👍

👎



I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

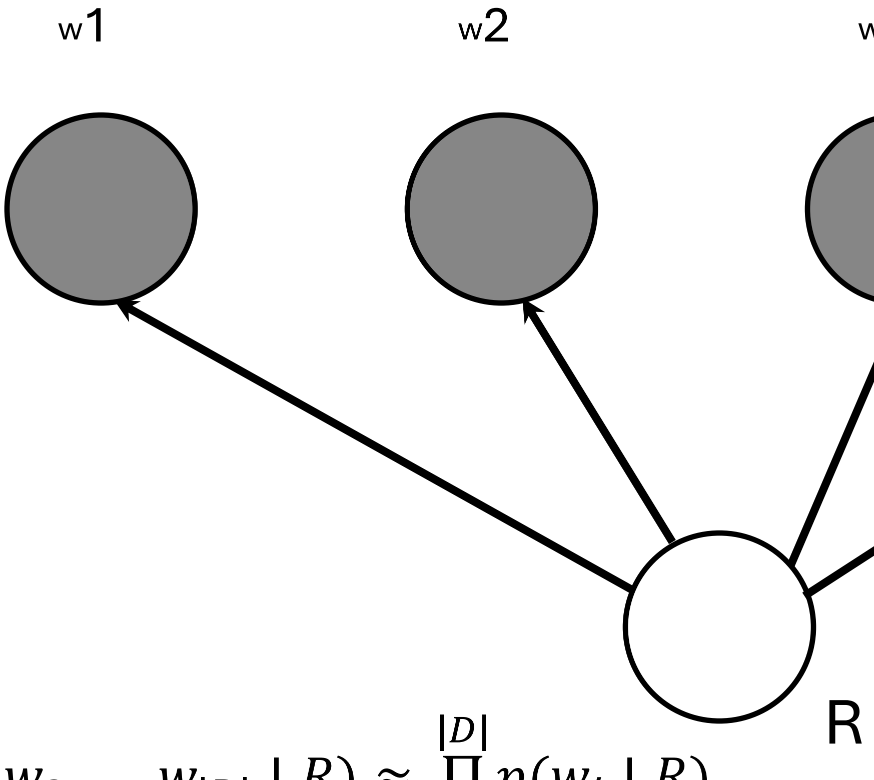
it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Figure 4.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Figure from Jurafsky and Martin

Naive Bayes Classifier

One variable per **token** in document



No dependencies among words!

$$p(w_1, w_2, \dots, w_{|D|} \mid R) \approx \prod_{i=1}^{|D|} p(w_i \mid R)$$

Naïve Bayes on Movie Reviews

- Train models for positive, negative
- For each review, find higher posterior
- Which word probability ratios are highest?

```
>>> classifier.show_most_informative_features(5)

classifier.show_most_informative_features(5)
Most Informative Features
  contains(outstanding) = True          pos : neg    =     14.1 : 1.0
       contains(mulan) = True           pos : neg    =      8.3 : 1.0
      contains(seagal) = True           neg : pos    =      7.8 : 1.0
  contains(wonderfully) = True          pos : neg    =      6.6 : 1.0
       contains(damon) = True           pos : neg    =      6.1 : 1.0
```

What’s Wrong With naïve Bayes?

- What happens when word dependencies are strong?
- What happens when some words occur only once?
- What happens when the classifier sees a new word?

We will talk more about naïve Bayes in the next lecture

ML for Naive Bayes

- Recall: p(+ | Damon movie)

  = p(Damon | +) p(movie | +) p(+)

- If corpus of positive reviews has 1000 words, and "Damon" occurs 50 times,

  p_{ML}(Damon | +) = ?

- If pos. corpus has "**Affleck**" **0 times**,

  p(+ | **Affleck** Damon movie) = ?

Out of Vocabulary (OOV)!

Estimation for Markov (n-gram) models

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word “bagel” occurs 400 times in a corpus of a million words

- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004

This may be a bad estimate for some other corpus

- But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Unigram example

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

Where $\mathrm{count}(w_{i-1}) = \sum_w \mathrm{count}(w_{i-1}, w)$

Total occurrences of w_{i-1} as a first word in any bigram

Intuition for bigram MLE

- The probability of a word following another word is just the **relative frequency** of that bigram.

- If "the cat" appears 50 times, and "the" appears 200 times in total, then:

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

- P(cat | the) = 50/200 = 0.25

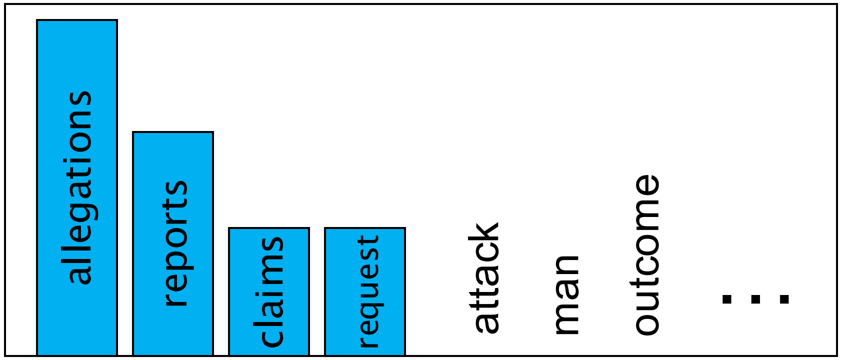
The intuition of smoothing

(Example modified from Dan Klein!)

When we have sparse statistics:

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

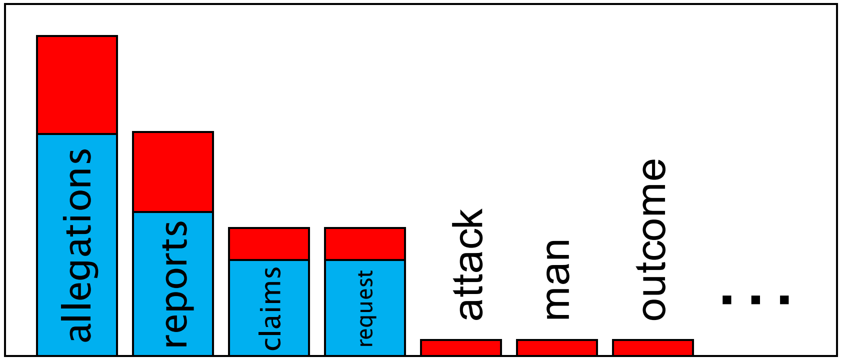
7 total



Steal probability mass to generalize better

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

7 total



Add-one estimation

- Also called **Laplace smoothing**
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

MLE estimate:

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

Add-1 estimate:

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

V be the vocabulary size

Compare with raw bigram counts

original

	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

add-1 smoothed

	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

Generalized Additive Smoothing

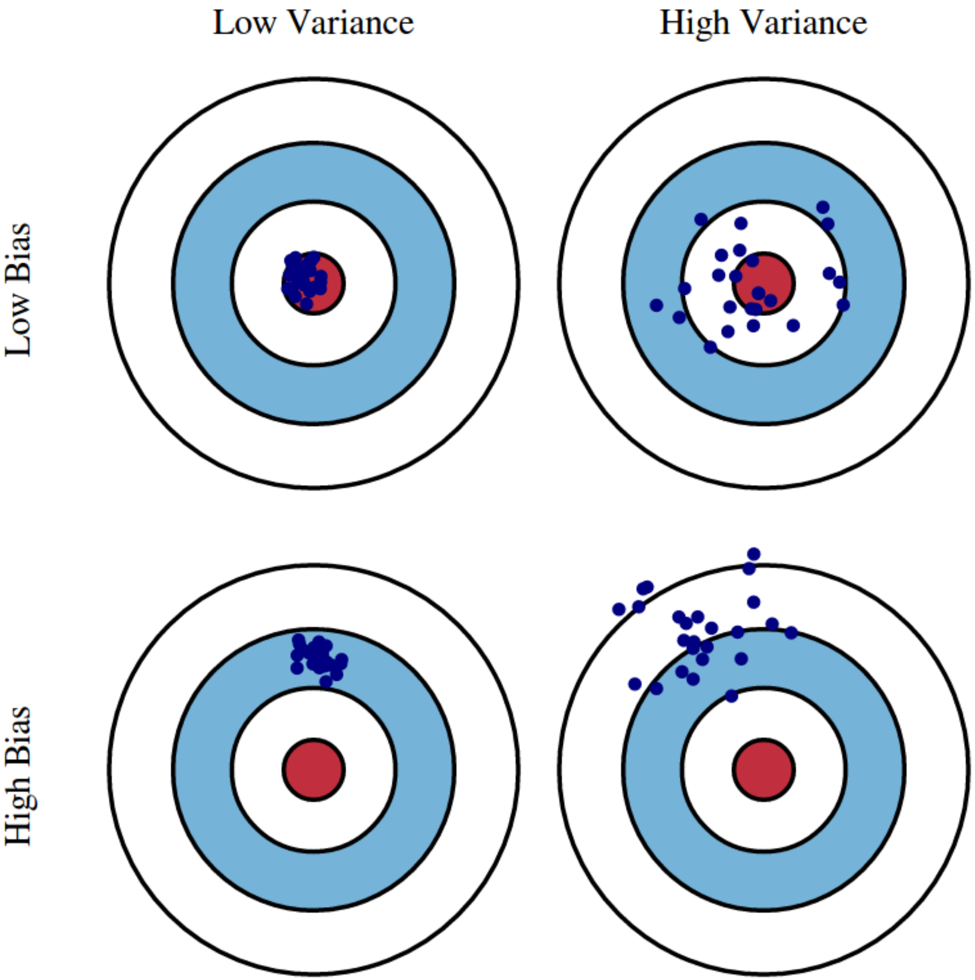
- Laplace add-one smoothing generally assigns *too much* probability to unseen words

- More common to use λ instead of 1:
  (Laplace is just a special case of where λ = 1)

What's the right λ?

$$
\begin{aligned}
p(w_3 \mid w_1, w_2) &= \frac{C(w_1, w_2, w_3) + \lambda}{C(w_1, w_2) + \lambda V} \\
&= \mu \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)} + (1 - \mu)\frac{1}{V} \\
\mu &= \frac{C(w_1, w_2)}{C(w_1, w_2) + \lambda V}
\end{aligned}
$$

interpolation


Low Variance
High Variance
Low Bias
High Bias

Bias-variance tradeoff

A little ML review, helpful for understanding MLE vs Smoothing

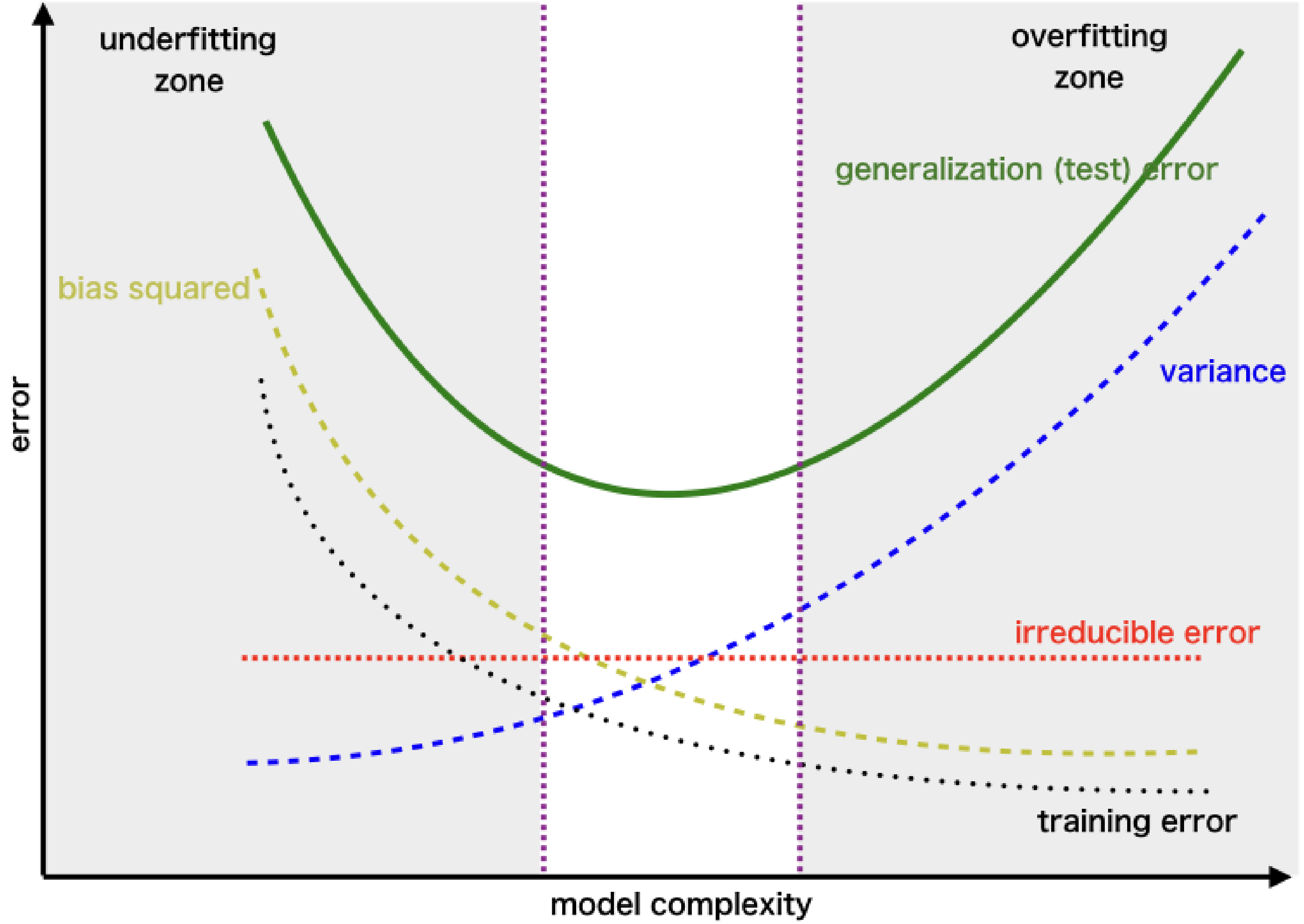
$$Expected\ Error = \ Bias^2 + Variance + Irreducible\ Error$$

- **Bias**: how far your model/estimator's average prediction is from the true value
  - High → model is too simple and not learning the pattern of the data → **underfitting**
  - Low → the model can capture the data well
- **Variance**: how much your model/estimator's prediction will fluctuate for different training sets
  - High → model is too sensitive to training data → **overfitting**
  - Low → predictions are stable; model can generalize well on different datasets
- You usually can't minimize both bias and variance at the same time

Bias-variance tradeoff

- Maximum likelihood is *asymptotically unbiased*: with a lot of data, the estimator learns the true value accurately
- Smoothing reduces variance and introduces some bias (bias-variance tradeoff!)
- High-variance classifiers may **overfit** the training data, performing poorly out of sample or unseen data
- Too much smoothing can lead to **underfitting**: as $\lambda \to \infty$ or $\mu \to 0$ we approach a uniform distribution, i.e., not learning the patterns from the training data

Bias-Variance Tradeoff



Backoff and Interpolation

Sometimes it helps to use a simpler model

- Condition on **less** context for contexts you know **less** about

Backoff:

- If enough evidence, use trigram $P(w_n|w_{n-2}w_{n-1})$
- If not, use bigram $P(w_n|w_{n-1})$
- Else unigram $P(w_n)$

Interpolation:

- mix unigram, bigram, trigram

Interpolation works better

Linear Interpolation

Simple interpolation

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

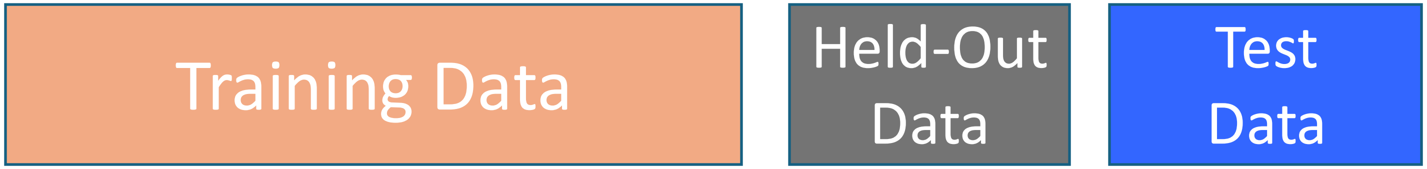
$$\sum_i \lambda_i = 1$$

Lambdas conditional on context:

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

How to set λs for interpolation?

1. Use a **held-out** corpus aka development data



Training Data

Held-Out Data

Test Data

Choose λs to maximize probability of held-out data:

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

2. K-fold cross-validation (jackknife)

3. Leave-one-out cross-validation

Summary: What to do if you never saw an n-gram in training

Smoothing: Pretend you saw every n-gram one (or k) times more than you did

- A blunt instrument (replacing a lot of zeros) but sometimes useful

Backoff: If you haven't seen the trigram, use the (weighted) bigram probability instead

- Weighting is messy; "stupid" backoff works fine at web-scale

Interpolation: (weighted) mix of trigram, bigram, unigram

- Usually the best! We also use interpolation to combine multiple LLMs

Logistics

- Assignment 1 is released
- Link to the recommended textbook is also online
- Office hours are online now. Attend if you need any help
- If you haven’t join Gradescope, make sure you do so. You will need this for submitting your coding assignment.
- Join Ed Discussion if you want to ask questions online.