NATURAL LANGUAGE PROCESSING
UNIT-II

N-gram Language Models: N-Grams, Evaluating Language Model, Sampling sentences from a language model, Sequence Labeling for Parts of Speech and Named Entities: Part-of-Speech Tagging, Named Entities and Named Entity Tagging

N-GRAM LANGUAGE MODELING
INTRODUCTION TO N-GRAM LANGUAGE MODELS

Predicting words

The water of Walden Pond is beautifully ...
blue
green
clear
*refrigerator
*that

Language Models
Task
Systems that can predict upcoming words
- Can assign a probability to each potential next word
- Can assign a probability to a whole sentence

Why word prediction?
- It's a helpful part of language tasks
- Grammar or spell checking
  Their are two midterms   Their There are two midterms
  Everything has improve   Everything has improve improved
- Speech recognition
  I will be back soonish   I will be bassoon dish

- It's how **large language models (LLMs)** work!

LLMs are **trained** to predict words
- Left-to-right (autoregressive) LMs learn to predict next word

LLMs **generate** text by predicting words
- By predicting the next word over and over again

Language Modeling (LM) more formally
Goal: compute the probability of a sentence or sequence of words W:
$P(W) = P(w_1,w_2,w_3,w_4,w_5\ldots w_n)$
Related task: probability of an upcoming word:
$P(w_5|w_1,w_2,w_3,w_4)$ or $P(w_n|w_1,w_2\ldots w_{n-1})$
An LM computes either of these:
$P(W)$ or $P(w_n|w_1,w_2\ldots w_{n-1})$

How to estimate these probabilities?

Could we just count and divide?

$$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully}) = \frac{C(\texttt{The water of Walden Pond is so beautifully blue})}{C(\texttt{The water of Walden Pond is so beautifully})}$$

No! Too many possible sentences!

We'll never see enough data for estimating these

How to compute P(W) or $P(w_n|w_1, \ldots w_{n-1})$???

How to compute the joint probability P(W):

P(The, water, of, Walden, Pond, is, so, beautifully, blue)

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

Reminder: The Chain Rule

Recall the definition of conditional probabilities

$P(B|A) = P(A,B)/P(A)$Rewriting: $P(A,B) = P(A)\,P(B|A)$

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

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

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

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

P("The water of Walden Pond") =

P(The) × P(water|The) × P(of|The water)

× P(Walden|The water of) × P(Pond|The water of Walden)

Markov Assumption

Simplifying assumption:

$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully}) \approx P(\text{blue}|\text{beautifully})$

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

Bigram Markov Assumption

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

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

More generally, we approximate each component in the product

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

Simplest case: Unigram model

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

Some automatically generated sentences from two different unigram models

> To him swallowed confess hear both . Which . Of save on trail for are ay device and rote life have
>
> Hill he late speaks ; or ! a more to leg less first you enter
> Months the my and issue of year foreign new exchange's September
> were recession exchange new endorsed a acquire to six executives

Bigram model

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

Some automatically generated sentences rom two different unigram models

Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

What means, sir. I confess she? then all sorts, he is trim, captain.
Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one gram 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

Problems with N-gram models

- N-grams can't handle **long-distance dependencies**:

"**The soups** that I made from that new cookbook I bought yesterday **were** amazingly delicious."

- N-grams don't do well at modeling new sequences with similar meanings

The solution: **Large language models**

- can handle much longer contexts
- because of using embedding spaces, can model synonymy better, and generate better novel strings

Why N-gram models?
A nice clear paradigm that lets us introduce many of the important issues for **large language models**

- **training** and **test** sets
- the **perplexity** metric
- **sampling** to generate sentences
- ideas like **interpolation** and **backoff**

INTRODUCTION TO N-GRAMS

N-Gram Models

1. P(office | about fifteen minutes from)

- An N-gram model uses only N −1 words of prior context
- Unigram: P(office)
- Bigram: P(office | from)
- Trigram: P(office | minutes from)
- Markov model and Language Model An N-gram model is an N −1-order Markov Model
- We can extend to trigrams, 4-grams, 5-grams ,N,,,,
- In general, an insufficient model of language:
- language has long-distance dependencies:
- "The **computer** which I had just put into the machine room on the fifth floor **crashed**."
- In most of the applications, we can get away with N-gram models

Estimating N-grams probabilities
Maximum Likelihood Estimate
Value that makes the observed data the "most probable"
P(wi |wi−1) = count(wi−1,wi) /count(wi−1)
P(wi |wi−1) = count(wi−1,wi) /count(wi−1)
P(wi |wi−1) = c(wi−1,wi) /c(wi−1)
<S>I am here</S>
<S>who am I</S>
<S>I would like to know</S>
Estimating bigrams
P(I|) =
P(|here) =
P(would | I) =
P(here | am) =
P(know | like) =

Estimating N-gram Probabilities

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

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

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

An example

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

$$P(\texttt{I}\,|\,\texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \tfrac{2}{3} = .67$$

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

More examples:

- Berkeley Restaurant Project sentences
- can you tell me about any good cantonese restaurants close by
- tell me about chez panisse
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Raw bigram counts

Out of 9222 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

Raw bigram probabilities

Normalize by unigrams:

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 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 do N-grams represent?

- 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

Dealing with scale in large n-grams

LM probabilities are stored and computed in log format, i.e. **log probabilities**
This avoids underflow from multiplying many small numbers

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

If we need probabilities we can do one exp at the end

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

Larger n-grams

4-grams, 5-grams
Large datasets of large n-grams have been released

- N-grams from Corpus of Contemporary American English (COCA) 1 billion words (Davies 2020)
- Google Web 5-grams (Franz and Brants 2006) 1 trillion words)
- Efficiency: quantize probabilities to 4-8 bits instead of 8-byte float

Newest model: infini-grams (∞-grams) (Liu et al 2024)

- No precomputing! Instead, store 5 trillion words of web text in **suffix arrays**. Can compute n-gram probabilities with any n!

N-gram LM Toolkits

SRILM

- http://www.speech.sri.com/projects/srilm/

KenLM

- https://kheafield.com/code/kenlm/

ESTIMATING N-GRAM PROBABILITIES EVALUATION AND PERPLEXITY

How to evaluate N-gram models
"Extrinsic (in-vivo) Evaluation"
To compare models A and B

1. Put each model in a real task
- Machine Translation, speech recognition, etc.
2. Run the task, get a score for A and for B
- How many words translated correctly
- How many words transcribed correctly
3. Compare accuracy for A and B

Intrinsic (in-vitro) evaluation

Extrinsic evaluation not always possible

- Expensive, time-consuming
- Doesn't always generalize to other applications

Intrinsic evaluation: perplexity

- Directly measures language model performance at predicting words.
- Doesn't necessarily correspond with real application performance
- But gives us a single general metric for language models
- Useful for large language models (LLMs) as well as n-grams

Training sets and test sets

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; different from training set.
    - Intuition: we want to measure generalization to unseen data
- An **evaluation metric** (like **perplexity**) tells us how well our model does on the test set.

Choosing training and test sets

- If we're building an LM for a specific task

  - The test set should reflect the task language we want to use the model for
- If we're building a general-purpose model
  - We'll need lots of different kinds of training data
  - We don't want the training set or the test set to be just from one domain or author or language.

Training on the test set

We can’t allow test sentences into the training set

- Or else the LM will assign that sentence an artificially high probability when we see it in the test set
- And hence assign the whole test set a falsely high probability.
- Making the LM look better than it really is

This is called “Training on the test set”

Bad science!

Dev sets

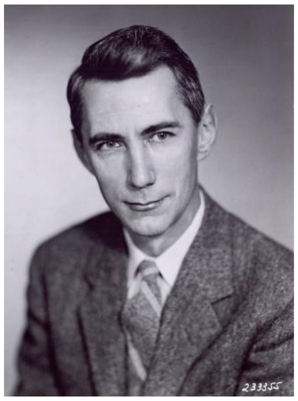
- If we test on the test set many times we might implicitly tune to its characteristics
  - Noticing which changes make the model better.
- So we run on the test set only once, or a few times
- That means we need a third dataset:
  - A **development test set** or, **devset**.
  - We test our LM on the devset until the very end
  - And then test our LM on the **test set** once
  -

Intuition of perplexity as evaluation metric: How good is our language model?

Intuition: A good LM prefers "real" sentences

- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?
-

Intuition of perplexity 2:
Predicting upcoming words


Claude Shannon

The Shannon Game: **How well can we predict the next word**?

- Once upon a ____
- That is a picture of a ____
- For breakfast I ate my usual ____

time	**0.9**
dream	**0.03**
midnight	**0.02**
…	
and	**1e-100**

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs

Intuition of perplexity 3: The best language model is one that best predicts the entire unseen test set

- We said: a good LM is one that assigns a higher probability to the next word that actually occurs.
- Let's generalize to all the words!
  - The best LM assigns high probability to the entire test set.
- When comparing two LMs, A and B
  - We compute P_A(test set) and P_B(test set)
  - The better LM will give a higher probability to (=be less surprised by) the test set than the other LM.

Intuition of perplexity 4: Use perplexity instead of raw probability

- Probability depends on size of test set
  - Probability gets smaller the longer the text
  - Better: a metric that is **per-word**, normalized by length
- **Perplexity** is the inverse probability of the test set, normalized by the number of words

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

Intuition of perplexity 5: the inverse

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

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

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)

Probability range is $[0,1]$, perplexity range is $[1,\infty]$

Minimizing perplexity is the same as maximizing probability

Intuition of perplexity 6: N-grams

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

Chain rule: $\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}}$

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

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

Intuition of perplexity 7:
Weighted average branching factor

Perplexity is also the **weighted average branching factor** of a language.
Branching factor: number of possible next words that can follow any word
Example: Deterministic language L = {red,blue, green}
Branching factor = 3 (any word can be followed by red, blue, green)
Now assume LM A where each word follows any other word with equal probability ⅓
Given a test set T = "red red red red blue"

$$\text{Perplexity}_A(T) = P_A(\text{red red red red blue})^{1/5} = ((⅓)^5)^{-1/5} = (⅓)^{-1} = 3$$

But now suppose red was very likely in training set, such that for LM B:

- P(red) = .8 p(green) = .1 p(blue) = .1

We would expect the probability to be higher, and hence the perplexity to be smaller:

$$\text{Perplexity}_B(T) = P_B(\text{red red red red blue})^{1/5}$$
$$= (.8 * .8 * .8 * .8 * .1)^{-1/5} = .04096^{-1/5} = .527^{-1} = 1.89$$

Holding test set constant:

Lower perplexity = better language model

Training 38 million words, test 1.5 million words, WSJ

<u>LANGUAGE MODELING</u>

Evaluation and Perplexity
Sampling and Generalization

The Shannon (1948) Visualization Method Sample words from an LM

Unigram:

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

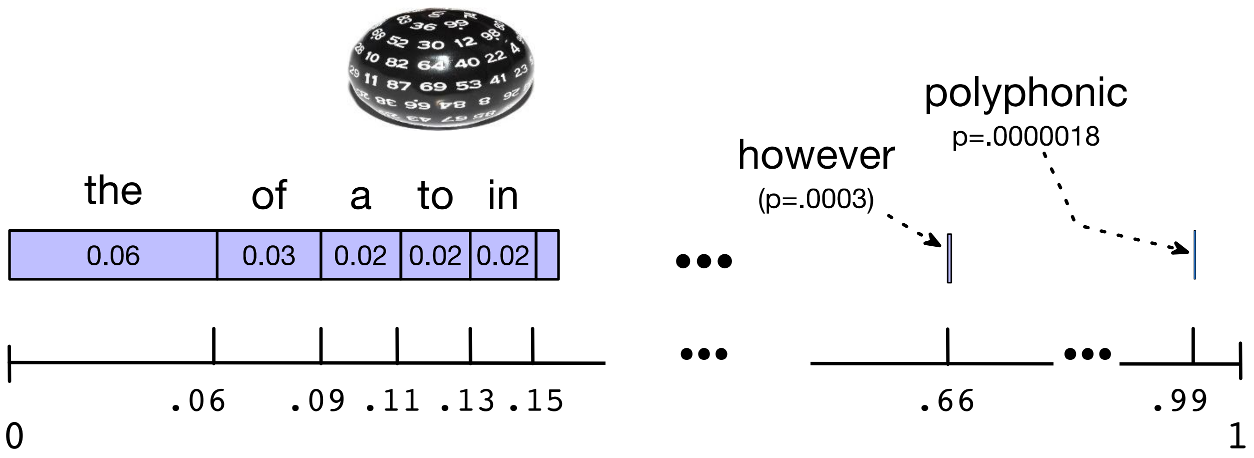
Bigram:

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

How Shannon sampled those words in 1948

"Open a book at random and select a letter at random on the page. This letter is recorded. The book is then opened to another page and one reads until this letter is encountered. The succeeding letter is then recorded. Turning to another page this second letter is searched for and the succeeding letter recorded, etc."

Sampling a word from a distribution



Visualizing Bigrams the Shannon Way

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

And so on until we choose </s>

Then string the words together

<s> I

I want

want to

to eat

eat Chinese

Chinese food

food </s>

I want to eat Chinese food

Note: there are other sampling methods

Used for neural language models

Many of them avoid generating words from the very unlikely tail of the distribution

We'll discuss when we get to neural LM decoding:

- Temperature sampling
- Top-k sampling
- Top-p sampling

Approximating Shakespeare

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

Shakespeare as 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)
- That sparsity is even worse for 4-grams, explaining why our sampling generated actual Shakespeare.

The Wall Street Journal is not Shakespeare

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

Can you guess the author? These 3-gram sentences are sampled from an LM trained on who?

1) 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 gram Brazil on market conditions

2) This shall forbid it should be branded, if renown made it empty.

3) "You are uniformly charming!" cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Choosing training data
If task-specific, use a training corpus that has a similar genre to your task.
- If legal or medical, need lots of special-purpose documents

Make sure to cover different kinds of dialects and speaker/authors.
- Example: *African-American Vernacular English (AAVE)*
    - One of many varieties that can be used by African Americans and others
    - Can include the auxiliary verb **finna** that marks immediate future tense:
    - "My phone finna die"

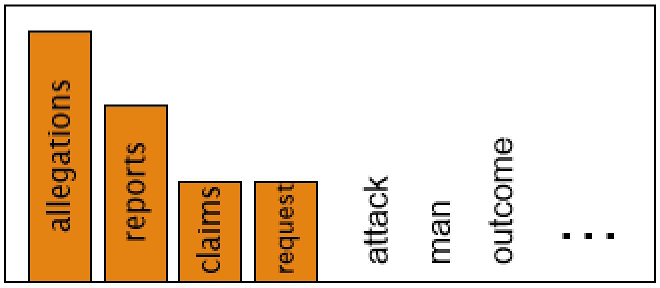
The perils of overfitting
N-grams only work well for word prediction if the test corpus looks like the training corpus
- But even when we try to pick a good training corpus, the test set will surprise us!
- We need to train robust models that generalize!

One kind of generalization: **Zeros**
- Things that don't ever occur in the training set
    - But occur in the test set

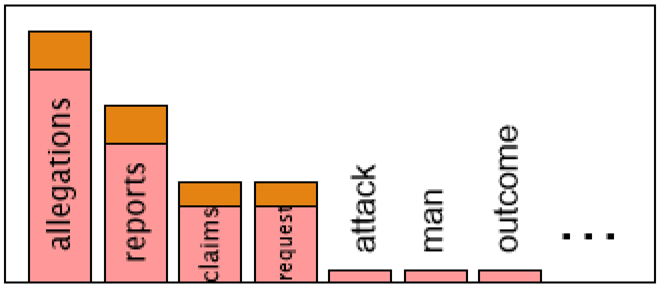
Zeros
Training set:
… ate lunch
… ate dinner
… ate a
… ate the
P("breakfast" | ate) = 0
Test set
… ate lunch
… ate breakfast

Zero probability bigrams
Bigrams with zero probability
- Will hurt our performance for texts where those words appear!
- And mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!
Sampling and Generalization
N-gram Language Modeling
Smoothing, Interpolation, and Backoff
The intuition of smoothing (from Dan Klein)
When we have 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



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:

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

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.

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

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

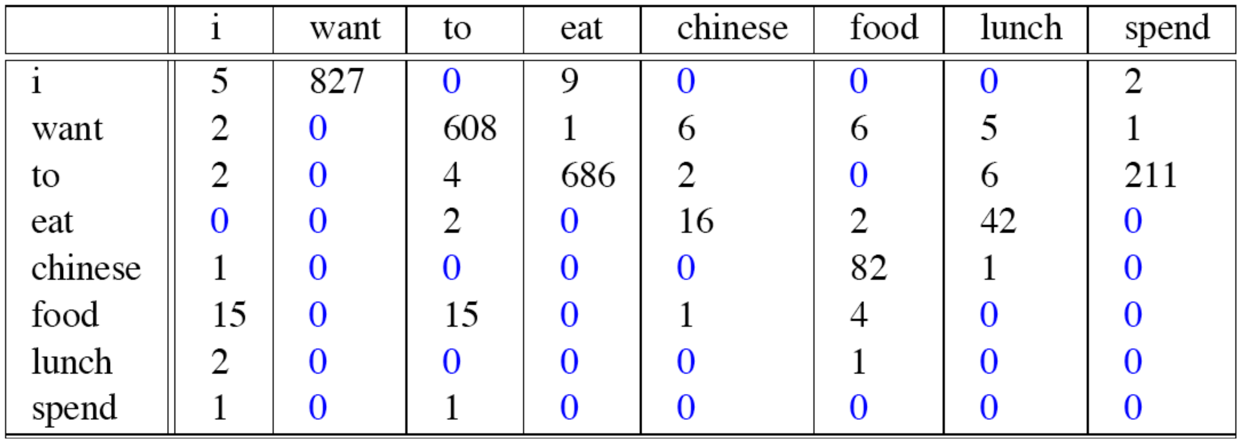
	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

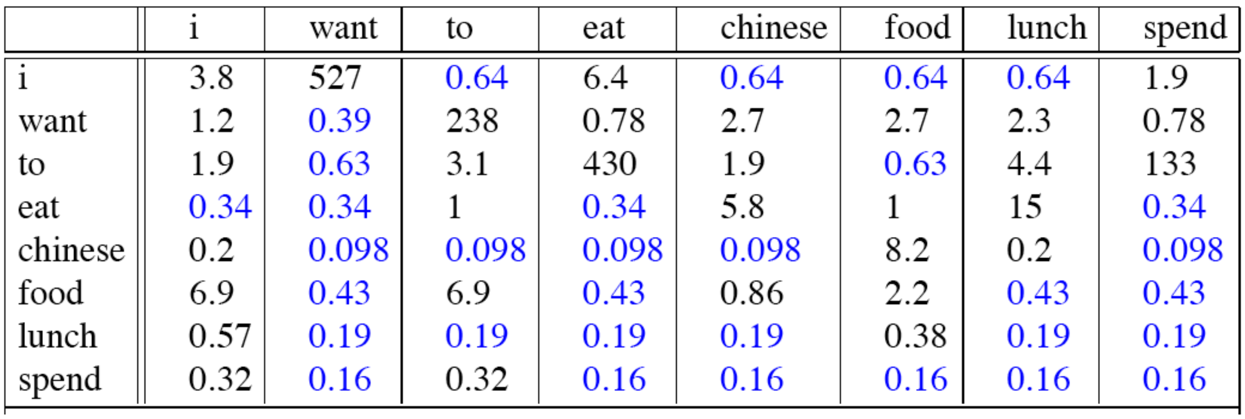
$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	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

Compare with raw bigram counts



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



	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

Add-1 estimation is a blunt instrument

So add-1 isn't used for N-grams:

Generally we use interpolation or backoff instead

But add-1 is used to smooth other NLP models

For text classification

In domains where the number of zeros isn't so huge.

Backoff and Interpolation

Sometimes it helps to use **less** context

Condition on less context for contexts you know less about

Backoff:

use trigram if you have good evidence,
otherwise bigram, otherwise unigram

Interpolation:

mix unigram, bigram, trigram
Interpolation works better

Linear Interpolation

Simple interpolation

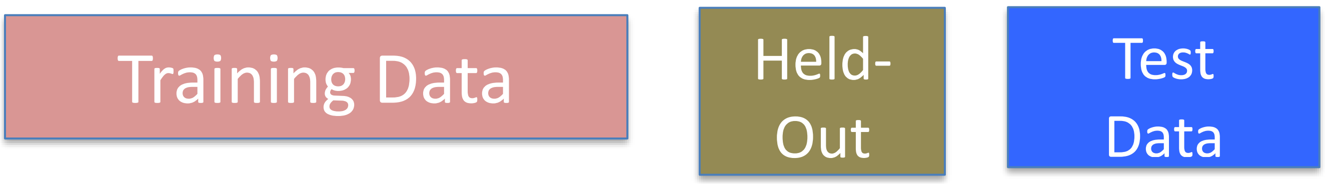
$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1})\\ &+\lambda_2 P(w_n|w_{n-1})\\ &+\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

Lambdas conditional on context:

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

How to set λs for interpolation?
Use a **held-out** corpus



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

Backoff
Suppose you want:
P(pancakes| delicious soufflé)
If the trigram probability is 0, use the bigram
P(pancakes| soufflé)
If the bigram probability is 0, use the unigram
P(pancakes)
Complication: need to discount the higher-order ngram so probabilities don't sum higher than 1 (e.g., Katz backoff)

Stupid Backoff:
Backoff without discounting (not a true probability)

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Questions to practice

1. In a corpus, you found that the word with rank 4th has a frequency of 250. What can be the best guess for the rank of a word with frequency 125?

frequency * rank =k [by Zipfs law]
250*4 = 125*r
r = 8

2. In the sentence, "In Delhi I took my hat off. But I can't put it back on.", total number of word tokens and word types are:14, 13.
   Solution: Here, the word "I" is repeated two times so type count is one less than token count.

3. If first corpus has $TTR1 = 0.06$ and second corpus has $TTR2 = 0.105$, where $TTR1$ and $TTR2$ represents type/token ratio in first and second corpus respectively, then

4. **Consider the following corpus C1 of 4 sentences. What is the total count of unique bi-grams for which the likelihood will be estimated? Assume we do not perform any pre-processing.**
   tomorrow is Sachin's birthday
   He loves cream chocolates
   he is also fond of sweet cake
   we will celebrate his birthday with sweet chocolate cake
   today is Sneha's birthday
   she likes ice cream
   she is also fond of cream cake
   we will celebrate her birthday with ice cream cake

Solution: Second corpus has more tendency to use different words. If TTR scores are higher then there is more tendency to use different words.
Detailed Solution:
Unique bi-grams are:

<s> tomorrow tomorrow is is Sachin's Sachin's birthday birthday <\s>
<s> he he loves loves cream cream chocolates chocolates <\s>
he is is also also fond fond of of sweet
cake <\s>
<s> we we will will celebrate celebrate his

his birthday birthday with with sweet chocolate cake

Unique bi-grams are:

<s> tomorrow	tomorrow is	is Sachin's	Sachin's birthday	birthday <\s>
<s> he	he loves	loves cream	cream chocolates	chocolates <\s>
he is	is also	also fond	fond of	of sweet
cake <\s>				
<s> we	we will	will celebrate	celebrate his	
his birthday	birthday with	with sweet	chocolate cake	

5. Consider the following corpus C3 of 3 sentences.
   there is a big garden
   children play in a garden
   they play inside beautiful garden

Calculate P(they play in a big garden) assuming a bi-gram language model.

P(they | <s>) = 1/3
P(play | they) = 1/1
P(in | play) = 1/2
P(a | in) = 1/1
P(big | a) = 1/2
P(garden | big) = 1/1
P(<\s>|garden) = 3/3
P(they play in a big garden) = 1/3 x 1/1 x 1/2 x 1/1 x 1/2 x 1/1 x 3/3 = 1/12

6. Given a corpus C2, the Maximum Likelihood Estimation (MLE) for the bigram "dried berries" is 0.45 and the count of occurrence of the word "dried" is 720. For the same corpus C2, the likelihood of "dried berries" after applying add-one smoothing is 0.05. What is the vocabulary size of C2?

$$P_{MLE}(berries|\ dried) = \frac{C(dried,\ berries)}{C(dried)}$$

0.45 = C(dried, berries) / 720

C(dried, berries) = 720*0.45 = 324

$$P_{Add-1}(berries|\ dried) = \frac{C(dried,\ berries)+1}{C(dried)+\ V}$$

0.05 = (324+1) / (720+V)

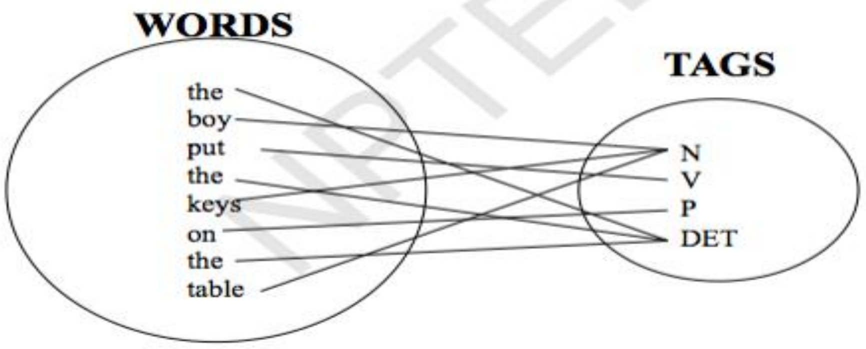
V=5780

POS TAGGING

PART-OF-SPEECH (POS) TAGGING

Task

Given a text of English, identify the parts of speech of each word



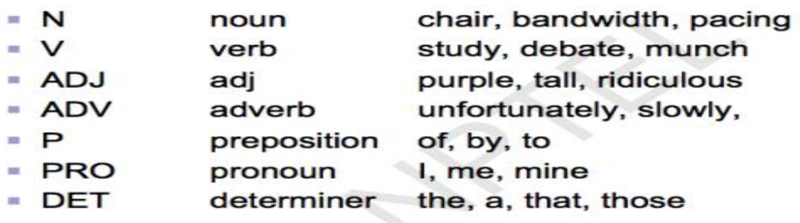
Parts of Speech: How many?

Open class words (content words)

nouns, verbs, adjectives, adverbs mostly content-bearing: they refer to objects, actions, and features in the world open class, since new words are added all the time

Closed class words

pronouns, determiners, prepositions, connectives, ... there is a limited number of these mostly functional: to tie the concepts of a sentence together



N	noun	chair, bandwidth, pacing
V	verb	study, debate, munch
ADJ	adj	purple, tall, ridiculous
ADV	adverb	unfortunately, slowly,
P	preposition	of, by, to
PRO	pronoun	I, me, mine
DET	determiner	the, a, that, those

POS tagging: Choosing a tagset

To do POS tagging, a standard set needs to be chosen

Could pick very coarse tagsets N, V, Adj, Adv

More commonly used set is finer grained, “UPenn TreeBank tagset”, 45 tags

UPenn TreeBank POS tag set

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... — -)
RP	Particle	*up, off*			

Using the UPenn tagset
Example Sentence
The grand jury commented on a number of other topics.
POS tagged sentence
The/DT grand/JJ jury/NN commmented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
Why is POS tagging hard?
Words often have more than one POS: back
The back door: back/JJ
On my back: back/NN
Win the voters back: back/RB
Promised to back the bill: back/VB
POS tagging problem
To determine the POS tag for a particular instance of a word
Ambiguous word types in the Brown Corpus
Ambiguity in the Brown corpus 40% of word tokens are ambiguous 12% of word types are ambiguous Breakdown of ambiguous word types:

Unambiguous (1 tag)	35,340
Ambiguous (2–7 tags)	4,100
2 tags	3,760
3 tags	264
4 tags	61
5 tags	12
6 tags	2
7 tags	1 ("still")

How bad is the ambiguity problem?
One tag is usually more likely than the others.

In the Brown corpus, race is a noun 98% of the time, and a verb 2% of the time
A tagger for English that simply chooses the most likely tag for each word can achieve good performance
Any new approach should be compared against the unigram baseline (assigning each token to its most likely tag)

Deciding the correct POS

- Can be difficult even for people
- Mrs./NNP Shaefer/NNP never/RB got/VBD around/RP to/TO joining/VBG.
- All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/IN the/DT corner/NN.
- Chateau/NNP Petrus/NNP costs/VBZ around/RB 2500/CD.

Relevant knowledge for POS tagging
The word itself

- Some words may only be nouns, e.g. arrow
- Some words are ambiguous, e.g. like, flies
- Probabilities may help, if one tag is more likely than another

Local context

- Two determiners rarely follow each other
- Two base form verbs rarely follow each other
- Determiner is almost always followed by adjective or noun

POS tagging: Two approaches
Rule-based Approach

- Assign each word in the input a list of potential POS tags
- Then winnow down this list to a single tag using hand-written rules

Statistical tagging

- Get a training corpus of tagged text, learn the transformation rules from the most frequent tags (TBL tagger)

Probabilistic: Find the most likely sequence of tags T for a sequence of words W
TBL Tagger

- Label the training set with most frequent tags
- The can was rusted.
- The/DT can/MD was/VBD rusted/VBD.
- Add transformation rules to reduce training mistakes
- MD →NN: DT_
- VBD→VBN: VBD_

Probabilistic Tagging:
Two different families of models

Problem at hand

- ✓ We have some data {(d, c)} of paired observations d and hidden classes c.
- ✓ Different instances of d and c
- ✓ Part-of-Speech Tagging: words are observed and tags are hidden.
- ✓ Text Classification: sentences/documents are observed and the

- ✓ category is hidden.
- ✓ Categories can be positive/negative for sentiments ..
- ✓ sports/politics/business for documents ...
- ✓ What gives rise to the two families?
- ✓ Whether they generate the observed data from hidden stuff or the hidden structure given the data?

GENERATIVE VS. CONDITIONAL MODELS
GENERATIVE (JOINT) MODELS

- ✓ Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: P(d, c) in terms of P(d|c)

e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.

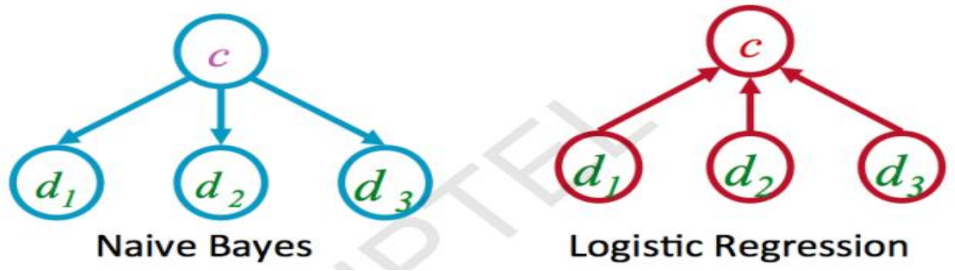
Discriminative (Conditional) Models

Take the data as given, and put a probability over hidden structure given the data: P(c|d)
e.g. Logistic regression, maximum entropy models, conditional random fields
SVMs, perceptron, etc. are discriminative classifiers but not directly probabilistic

Probabilistic Tagging:
Two different families of models
Problem at hand
We have some data {(d, c)} of paired observations d and hidden classes c.
Different instances of d and c
Part-of-Speech Tagging: words are observed and tags are hidden.
Text Classification: sentences/documents are observed and the
category is hidden.
Categories can be positive/negative for sentiments ..
sports/politics/business for documents ...
What gives rise to the two families?
Whether they generate the observed data from hidden stuff or the hidden structure given the data?
Generative vs. Conditional Models

> Generative (Joint) Models
> Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: P(d, c) in terms of P(d|c)
> e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.
> Discriminative (Conditional) Models
> Take the data as given, and put a probability over hidden structure given the data: P(c|d)
> e.g. Logistic regression, maximum entropy models, conditional random fields
> SVMs, perceptron, etc. are discriminative classifiers but not directly probabilistic

Joint vs. conditional likelihood
A joint model gives probabilities P(d, c) and tries to maximize this joint likelihood.
A conditional model gives probabilities P(c|d), taking the data as given and modeling only the conditional probability of the class.

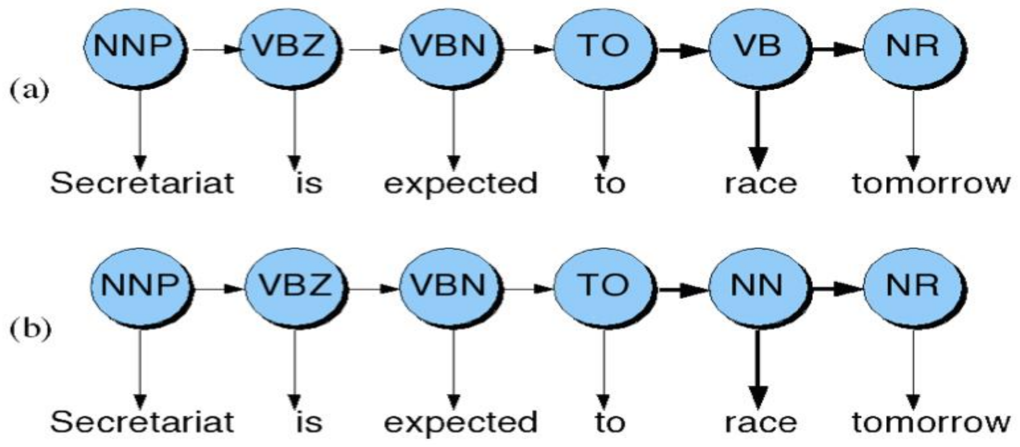


Hidden Markov Models for POS Tagging

Computing the probability values

Tag Transition probabilities

Disambiguating "race"



Difference in probability due to
P(VB/TO) vs. P(NN/TO)
P(race/VB) vs. P(race/NN)
P(NR/VB) vs. P(NR/NN)
After computing the probabilities
P(NN/TO)P(NR/NN)P(race/NN)=0.0047 X 0.0012 X 0.00057=0.00000000032
P(VB/TO)P(NR/VB)P(race/VB)=0.83 X 0.0027 X 0.00012=0.00000027

<u>HIDDEN MARKOV MODELS</u>

- Tag Transition probabilities p(ti/ti-1)
- Word Likelihood probabilities (emissions) p(wi/ti)
- What we have described with these probabilities is a hidden markov
- model.
- Let us quickly introduce the Markov Chain, or observable Markov Model.

Markov Chain = First-order Markov Model

Weather example

Three types of weather: sunny, rainy, foggy

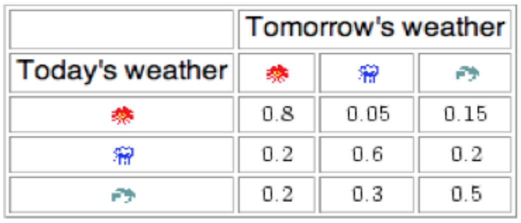
qn: variable denoting the weather on the nth day

We want to find the following conditional probabilities:

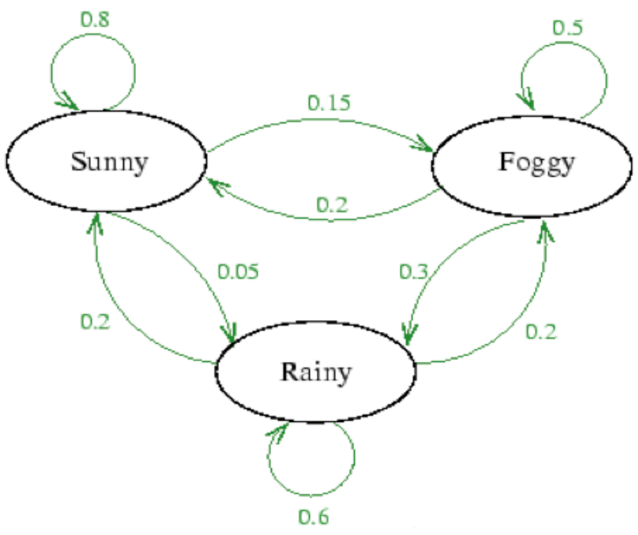
$$P(q_n | q_{n-1}, \cancel{q_{n-2}, \ldots, q_1}) = P(q_n | q_{n-1})$$

First-order Markov Assumption

Table 1: Probabilities $p(q_{n+1}|q_n)$ of tomorrow's weather based on today's weather



	Tomorrow's weather		
Today's weather	☀	🌧	🌫
☀	0.8	0.05	0.15
🌧	0.2	0.6	0.2
🌫	0.2	0.3	0.5



Using Markov Chain

Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

P(q2 = sunny;q3 = rainyI q1 = sunny)

= P(q3 = rainyjq2 = sunny;q1 = sunny)P(q2 = sunnyIq1 = sunny)

= P(q3 = rainyjq2 = sunny)P(q2 = sunnyIq1 = sunny)

= 0.05 X 0.8

= 0.04

For Markov chains, the output symbols are the same as the states

'sunny' weather is both observable and state

But in POS tagging

The output symbols are words

But the hidden states are POS tags

A Hidden Markov Model is an extension of a Markov chain in which the output symbols are not the same as the states

We don't know which state we are in

Elements of an HMM model

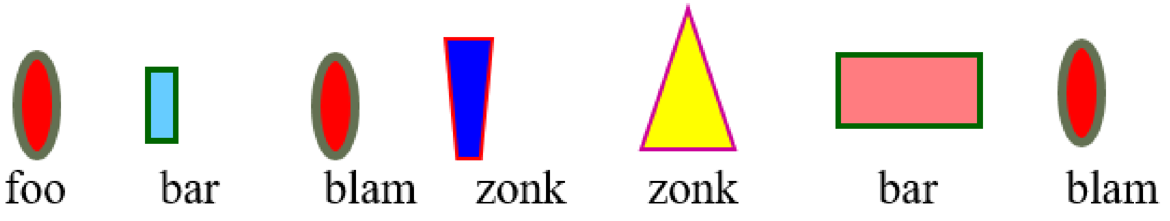
- A set of states (here: the tags)
- An output alphabet (here: words)
- Initial state (here: beginning of sentence)

State transition probabilities (here $p(t_n|t_{n-1})$)

Symbol emission probabilities (here $p(w_i|t_i)$)

Sequence Labeling Problem

- Many NLP problems can viewed as sequence labeling.
- Each token in a sequence is assigned a label.
- Labels of tokens are dependent on the labels of other tokens in the sequence, particularly their neighbors (not i.i.d).



Information Extraction

- Identify phrases in language that refer to specific types of entities and relations in text.
- Named entity recognition is task of identifying names of people, places, organizations, etc. in text.

people organizations places

- ◦ Michael Dell is the CEO of Dell Computer Corporation and lives in Austin Texas.
- ◦ Extract pieces of information relevant to a specific application, e.g. used car ads:

make model year mileage price

- ◦ For sale, 2002 Toyota Prius, 20,000 mi, $15K or best offer. Available starting July 30, 2006.

Semantic Role Labeling

For each clause, determine the semantic role played by each noun phrase that is an argument to the verb.

agent patient source destination instrument

- ◦ John drove Mary from Austin to Dallas in his Toyota Prius.
- ◦ The hammer broke the window.

Also referred to a “case role analysis,” “thematic analysis,” and “shallow semantic parsing”

Bioinformatics

Sequence labeling also valuable in labeling genetic sequences in genome analysis.

extron intron

- ◦ AGCTAACGTTCGATACGGATTACAGCCT

Problems with Sequence Labeling as Classification

Not easy to integrate information from category of tokens on both sides.

Difficult to propagate uncertainty between decisions and “collectively” determine the most likely joint assignment of categories to all of the tokens in a sequence.

Probabilistic Sequence Models

Probabilistic sequence models allow integrating uncertainty over multiple, interdependent classifications and collectively determine the most likely global assignment.

Two standard models

- ◦ Hidden Markov Model (HMM)
- ◦ Conditional Random Field (CRF)

Markov Model / Markov Chain

A finite state machine with probabilistic state transitions.

Makes Markov assumption that next state only depends on the current state and independent of previous history.

Formal Definition of an HMM

A set of $N+2$ states $S=\{s_0, s_1, s_2, \ldots s_N, s_F\}$

- Distinguished start state: s_0
- Distinguished final state: s_F

A set of M possible observations $V=\{v_1, v_2 \ldots v_M\}$

A state transition probability distribution $A=\{a_{ij}\}$

$$a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i) \qquad 1 \le i, j \le N \text{ and } i = 0, j = F$$

$$\sum_{j=1}^{N} a_{ij} + a_{iF} = 1 \quad 0 \le i \le N$$

Observation probability distribution for each state j $B=\{b_j(k)\}$

Total parameter set $\lambda=\{A,B\}$

$$b_j(k) = P(v_k \text{ at } t \mid q_t = s_j) \quad 1 \le j \le N \quad 1 \le \mathrm{k} \le \mathrm{M}$$

HMM Generation Procedure

To generate a sequence of T observations: $O = o_1\, o_2 \ldots o_T$

Set initial state $q_1=s_0$

```
For $t$ = 1 to $T$
    Transit to another state $q_{t+1}=s_j$ based on transition
      distribution $a_{ij}$ for state $q_t$
    Pick an observation $o_t=v_k$ based on being in state $q_t$ using
      distribution $b_{q_t}(k)$
```

Three Useful HMM Tasks

- Observation Likelihood: To classify and order sequences.
- Most likely state sequence (Decoding): To tag each token in a sequence with a label.
- Maximum likelihood training (Learning): To train models to fit empirical training data.

HMM: Observation Likelihood

Given a sequence of observations, O, and a model with a set of parameters, λ, what is the probability that this observation was generated by this model: $P(O|\lambda)$?

Allows HMM to be used as a language model: A formal probabilistic model of a language that assigns a probability to each string saying how likely that string was to have been generated by the language.

Useful for two tasks:

- Sequence Classification
- Most Likely Sequence

Sequence Classification

Assume an HMM is available for each category (i.e. language).

What is the most likely category for a given observation sequence, i.e. which category's HMM is most likely to have generated it?

Used in speech recognition to find most likely word model to have generate a given sound or phoneme sequence.

Most Likely Sequence
Of two or more possible sequences, which one was most likely generated by a given model?
Used to score alternative word sequence interpretations in speech recognition.

HMM: Observation Likelihood
Naïve Solution
Consider all possible state sequences, Q, of length T that the model could have traversed in generating the given observation sequence.
Compute the probability of a given state sequence from A, and multiply it by the probabilities of generating each of given observations in each of the corresponding states in this sequence to get $P(O,Q|\lambda) = P(O|Q,\lambda)\,P(Q|\lambda)$.
Sum this over all possible state sequences to get $P(O|\lambda)$.
Computationally complex: $O(TN^T)$.
Efficient Solution
Due to the Markov assumption, the probability of being in any state at any given time t only relies on the probability of being in each of the possible states at time $t-1$.
Forward Algorithm: Uses dynamic programming to exploit this fact to efficiently compute observation likelihood in $O(TN^2)$ time.

- Compute a *forward trellis* that compactly and implicitly encodes information about all possible state paths.

HMM: Most Likely State Sequence
Efficient Solution
Obviously, could use naïve algorithm based on examining every possible state sequence of length T.
Dynamic Programming can also be used to exploit the Markov assumption and efficiently determine the most likely state sequence for a given observation and model.
Standard procedure is called the Viterbi algorithm (Viterbi, 1967) and also has $O(N^2T)$ time complexity.

Viterbi Scores

Recursively compute the probability of the most likely subsequence of states that accounts for the first t observations and ends in state s_j.

$$v_t(j) = \max_{q_0, q_1, \ldots, q_{t-1}} P(q_0, q_1, \ldots, q_{t-1},\ o_1, \ldots, o_t,\ q_t = s_j \mid \lambda)$$

- Also record "backpointers" that subsequently allow backtracing the most probable state sequence.
  - $bt_t(j)$ stores the state at time t-1 that maximizes the probability that system was in state s_j at time t (given the observed sequence).

Computing the Viterbi Scores
Initialization

$$v_1(j) = a_{0j} b_j(o_1) \quad 1 \le j \le N$$

Recursion

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \quad 1 \le j \le N, \;\; 1 < t \le T$$

Termination

$$P* = v_{T+1}(s_F) = \max_{i=1}^{N} v_T(i) a_{iF}$$

Analogous to Forward algorithm except take *max* instead of sum
Computing the Viterbi Backpointers
Initialization

$$bt_1(j) = s_0 \quad 1 \le j \le N$$

Recursion

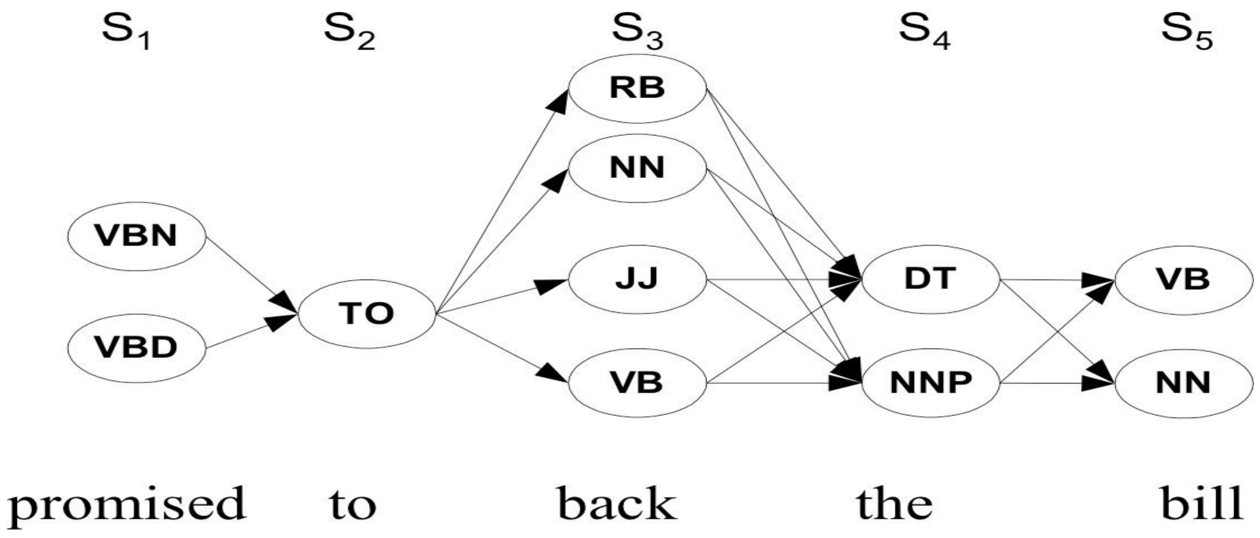
$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \quad 1 \le j \le N, \;\; 1 \le t \le T$$

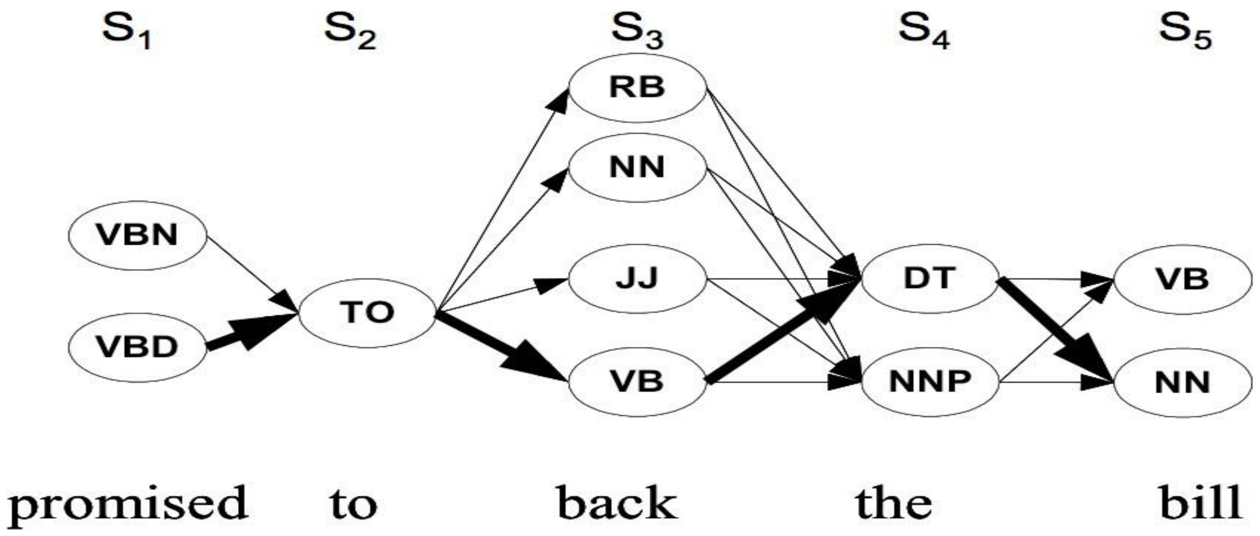
Termination

$$q_T* = bt_{T+1}(s_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) a_{iF}$$

Final state in the most probable state sequence. Follow
backpointers to initial state to construct full sequence.

Viterbi Decoding for HMM, Parameter Learning
Walking through the states: best path





Finding the best path: Viterbi Algorithm

Intuition

Optimal path for each state can be recorded. We need

- Cheapest cost to state j at step s: $\delta_j(s)$
- Backtrace from that state to best predecessor $\psi_j(s)$

Computing these values

- $\delta_j(s + 1) = \max_{1\leq i\leq N} \delta_i(s)p(t_j|t_i)p(w_{s+1}|t_j)$ $\psi_j(s + 1) = \text{argmax}_{1\leq i\leq N} \delta_i(s)p(t_j|t_i)p(w_{s+1}|t_j)$

Best final state is $argmax_{1\leq i\leq N}\, \delta_i(|S|)$, we can backtrack from there

Practice Question

- Suppose you want to use a HMM tagger to tag the phrase, “the light book”, where we have the following probabilities:
- P(the|Det) = 0.3, P(the|Noun) = 0.1, P(light|Noun) = 0.003, P(light|Adj) = 0.002, P(light|Verb) = 0.06, P(book|Noun) = 0.003, P(book|Verb) = 0.01
- P(Verb|Det) = 0.00001, P(Noun|Det) = 0.5, P(Adj|Det) = 0.3,
- P(Noun|Noun) =0.2, P(Adj|Noun) = 0.002, P(Noun|Adj) = 0.2,
- P(Noun|Verb) = 0.3, P(Verb|Noun) = 0.3, P(Verb|Adj) = 0.001,

- P(Verb|Verb) = 0.1
  Work out in details the steps of the Viterbi algorithm. You can use a Table to show the steps. Assume all other conditional probabilities, not mentioned to be zero. Also, assume that all tags have the same probabilities to appear in the beginning of a sentence.

Learning the Parameters
Two Scenarios

- A labeled dataset is available, with the POS category of individual words in a corpus
- Only the corpus is available, but not labeled with the POS categories

Methods for these scenarios

- For the first scenario, parameters can be directly estimated using maximum likelihood estimate from the labeled dataset
- For the second scenario, *Baum-Welch Algorithm* is used to estimate the parameters of the hidden markov model.

Baum Welch Algorithm
Uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden markov model
Parameters of HMM
Let X_t be the random variable denoting hidden state at time t, and Y_t be the observation variable at time T. HMM parameters are given by $\theta = (A, B, \pi)$ where

- $A = \{a_{ij}\} = P(X_t = j|X_{t-1} = i)$ is the state transition matrix
- $\pi = \{\pi_i\} = P(X_1 = i)$ is the initial state distribution
- $B = \{b_j(y_t)\} = P(Y_t = y_t|X_t = j)$ is the emission matrix

Given observation sequences $Y = (Y_1 = y_1, Y_2 = y_2, \ldots , Y_T = y_T)$, the algorithm tries to find the parameters θ that maximise the probability of the observation.
The Algorithm
The basic idea is to start with some random initial conditions on the parameters θ, estimate best values of state paths X_t using these, then
re-estimate the parameters θ using the just-computed values of X_t, iteratively.
Intuition

- Choose some initial values for $\theta = (A, B, \pi)$.
- *Repeat the following step until convergence:*
- Determine probable (state) paths $\ldots X_{t-1} = i, X_t = j \ldots$
- Count the expected number of transitions a_{ij} as well as the expected number of times, various emissions $b_j(y_t)$ are made
- Re-estimate $\theta = (A, B, \pi)$ using a_{ij} and $b_j(y_t)$s.

A forward-backward algorithm is used for finding probable paths.
Forward-Backward Algorithm
Forward Procedure
$\alpha_i(t) = P(Y_1 = y_1, \ldots , Y_t = y_t, X_t = i|\theta)$ be the probability of seeing $y_1, \ldots , y_t$ and being in state i at time t. Found recursively using:

$$\boldsymbol{\alpha_i(1) = \pi_i b_i(y_1)}$$
$$\boldsymbol{\alpha_i(t+1) = b_j(y_{t+1}) X_{N_{i=1}}^{\alpha_i(t)a_{ij}}}$$

Backward Procedure

$\beta_i(t) = P(Y_{t+1} = y_{t+1},..., Y_T = y_T|X_t = i, \theta)$ be the probability of ending partial sequence $y_{t+1},..., y_T$ given starting state i at time t. $\beta_i(t)$ is computed recursively as:
$\beta_i(T) = 1$
$\beta\ (t)$ =XN$\beta\ (t + 1)a_{ij}b_j(y_{t+1})\ j=1$

Finding probabilities of paths
We compute the following variables:
Probability of being in state i at time t given the observation Y and parameters θ

$$\gamma_i(t) = P(X_t = i|Y, \vartheta) = \frac{\alpha_i(t)\beta_i(t)}{\cdot\ \sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

Probability of being in state i and j at time t and $t + 1$ respectively given the observation Y and parameters θ

$$\zeta_{ij}(t) = P\ \ (X_t = i, X_{\ t+1}\ = j|Y, \vartheta) = \frac{\alpha_i(t)a_{ij}\beta_j(t + 1)b_j(y_{t+1})}{\cdot\ \sum_{i=1}^{N} \cdot\ \sum_{j=1}^{N} \alpha_i(t)a_{ij}\beta_j(t + 1)b_j(y_{t+1})}$$

Updating the parameters

- $\pi_i = \gamma_i(1)$, expected number of times state i was seen at time 1
- $a_{ij} = \frac{\sum_{t=1}^{T}\zeta_{ij}(t)}{\sum_{t=1}^{T}\gamma_{i(t)}}$ expected number of transitions from state i to state j, compared to the total number of transitions away from state i
- $b_i(v_k) = \frac{\sum_{t=1}^{T}1_{yt=vk}\gamma_i(t)}{\sum_{t=1}^{T}\gamma_i(t)}$ with $1_{yt\ =vk}$ being an indicator function, is the expected number of times the output observations are v_k while being in state i compared to the expected total number of times in state i.