Natural Language Processing

Lecture 2 - N-gram Language Models & Hidden Markov Models

April 2025

Eyal Ben-David

(DON’T FORGET RECORDING!!)

Course Logistics & updates

- First assignment will be released next week
  - Find your partner

- Students that are not listed to the course - come talk to next week:)

Recap - How do we do NLP?

- 1950 -- ~1990s ⟶ Write many rules
- 1990 -- ~2000 ⟶ Corpus-based statistics
- 2000 -- 2013 ⟶ Supervised machine learning
- 2013 -- today ⟶ “deep learning!#@$!@”

Recap - How do we do NLP?

- 1950 -- ~1990s ⟶ Write many rules
- 1990 -- ~2000 ⟶ Corpus-based statistics
- 2000 -- 2013 ⟶ Supervised machine learning
- 2013 -- today ⟶ “deep learning!#@$!@”

Recap - When should we learn?

- Learning typically works well when we have:
  1. a sufficient understanding the general *structure* of the problem
  2. lots and lots (and lots) of *data*

Today's Agenda

- **Language Modeling**
- N-gram Models
- The Tagging Problem
- Hidden Markov Models

Language Modeling

Language Modeling

Goal:

Assign a probability to a given sentence or sequence of words.

P(“Insanity is doing the same thing over and over again and expecting different results”) = ?

Language Modeling

Goal:

Assign a probability to a given sentence or sequence of words.

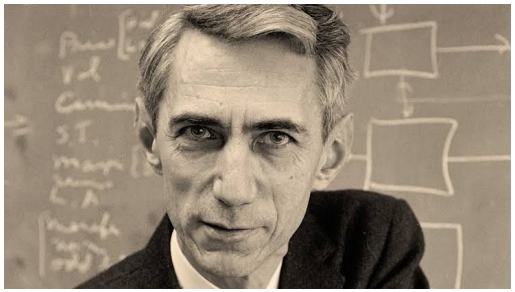
P(“Insanity is doing the same thing over and over again and expecting different results”) = ?

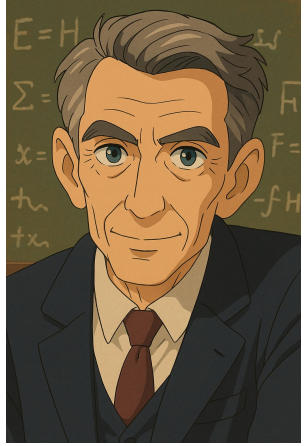
Another way of thinking about this:

Given a sequence of words, what is likely to be the next word?

P(“expecting” | “Insanity is doing the same thing over and over again and… ”) = ?

Shannon's "guessing game" (1951)





A 2 participants game

The first participant writes a prefix with empty boxes as suffix:

The dog ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

The second participant tries to guess the next letter within minimum guesses

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

Shannon's "guessing game" (1951)

A 2 participants game

The first participant writes a prefix with empty boxes as suffix:

The dog ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

The second participant tries to guess the next letter within minimum guesses

The dog *c* *h* *a* *s* *e* *d* *-* *t* *h* *e* *-* *m* *o* *u* *s* *e*

An AI-Complete Game

- Humans are great in this game
  - Without training
  - It is hard (for us) to get better at it

- Machines are (were) not that good at it
  - Practicing makes them better and better (more data)

An AI-Complete Game

Playing the game "at human level" would mean solving every other problem of AI and exhibiting human intelligence. *(taken from a blog written by Yoav Goldberg)*

Why is that? Think about it:

- The machine can complete any text prefix (!!)
- Including very long prefixes
- Including dialogues
- Including conversations
- Including description of tasks expressed in human language
- Including question (the suffix is the answer)
- Including math equations and proofs
- Including philosophy
- Etc.

An AI-Complete Game

Playing the game "at human level" would mean solving every other problem of AI and exhibiting human intelligence (Y. Goldberg, blog)

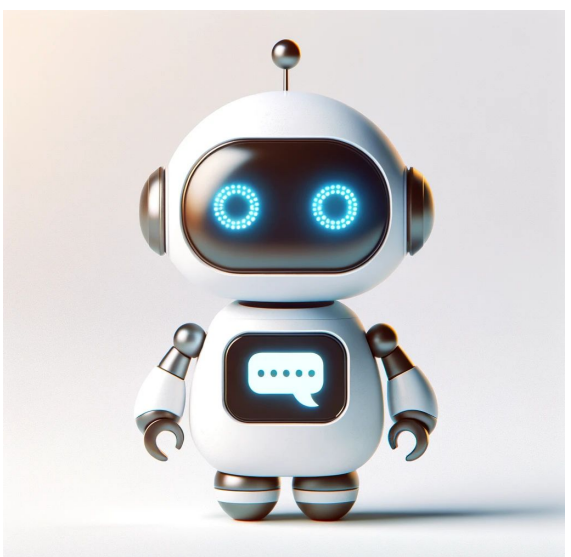
Being good at this game requires:

- Understand text
- Understand different situation described in text
- Imagine yourself in the situation
- Respond

An AI-Complete Game

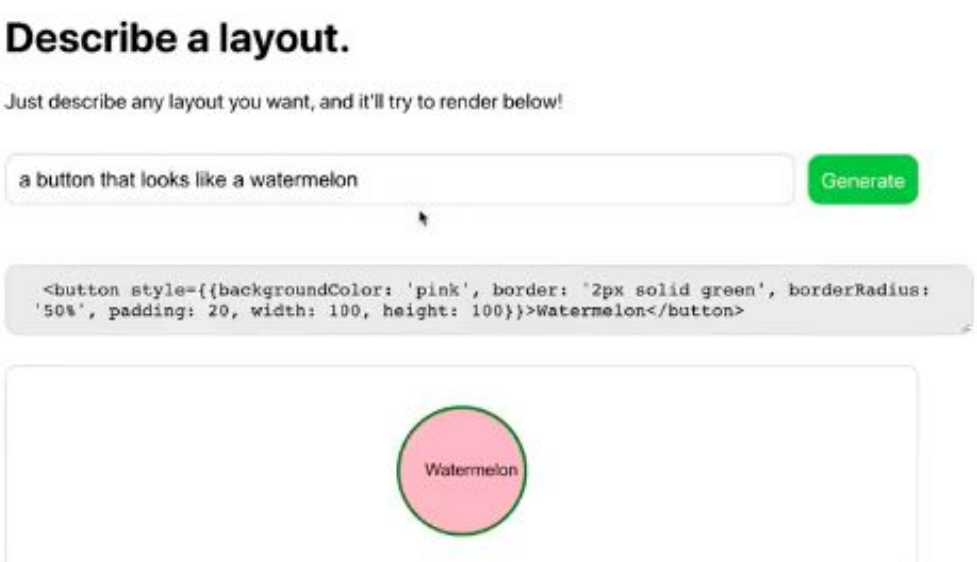
“Accurate token prediction is not just about predicting the next word or character; it involves understanding the underlying reality that led to the creation of that token”

Ilya Sutskev, 2023



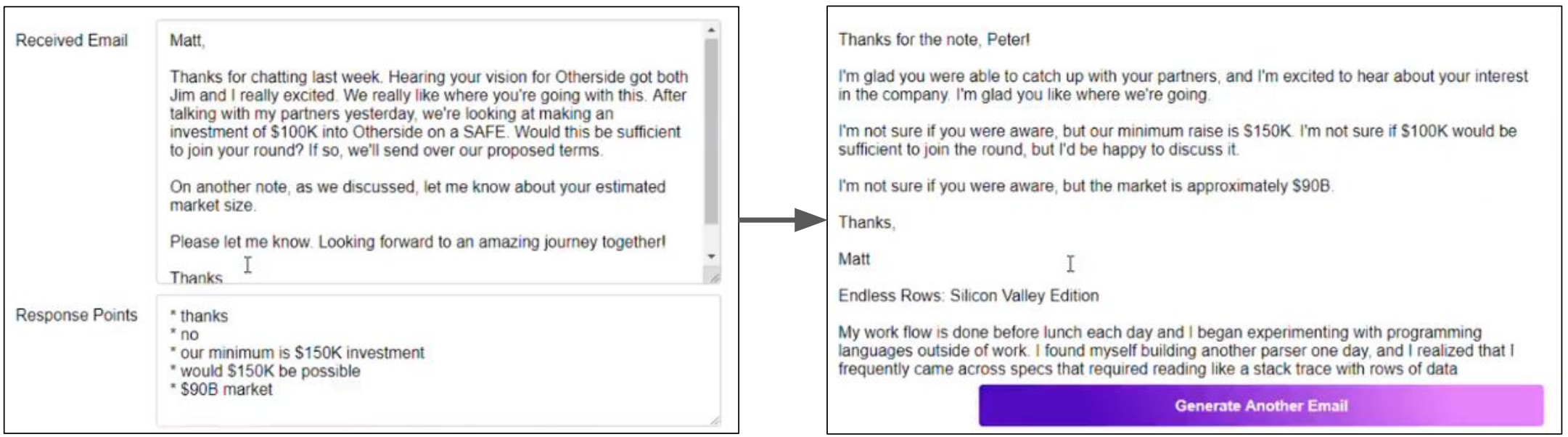
What can Language Models Do?

Generate a layout



What can Language Models Do?

Write my Email



What can Language Models Do?

Welcome student to the NLP class

(We asked GPT3** to write)
“The opening words of an NLP course:”

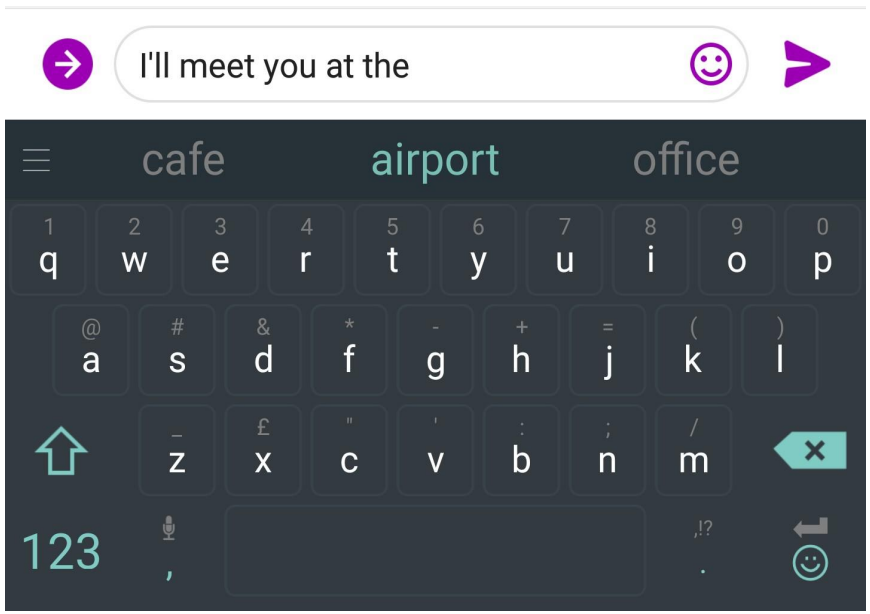
Welcome to our NLP course. In this course, we'll be learning about the principles and techniques of NLP, and how to apply them in our lives. We'll be covering a lot of ground in this course, so let's get started.

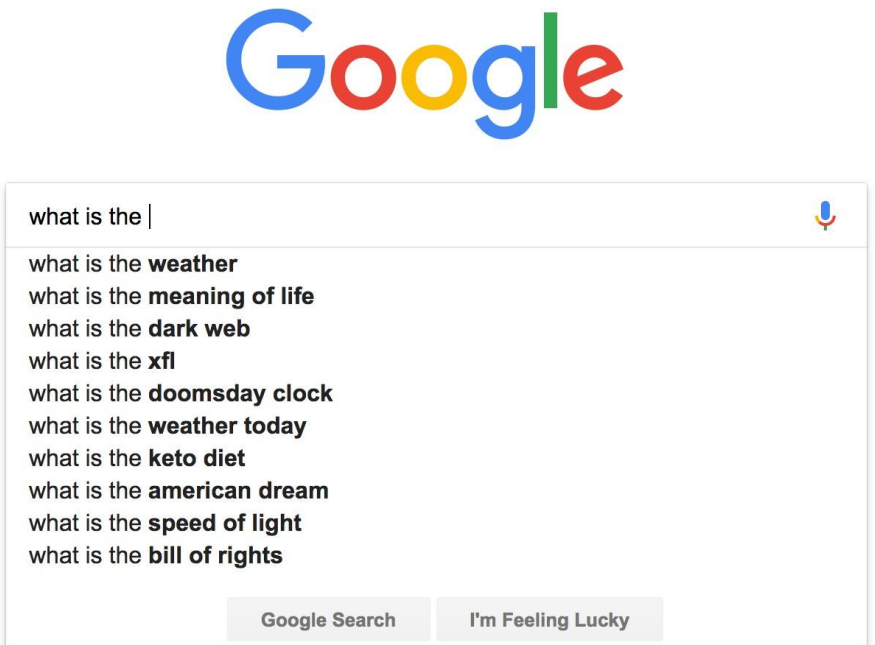
In this first lecture, we'll be introducing the basic concepts of NLP. We'll be talking about what NLP is, its history, and its main goals. We'll also be discussing the different components of NLP, and how they work together. Finally, we'll be giving you an overview of the different NLP techniques that we'll be covering in this course.

We hope you enjoy the course!

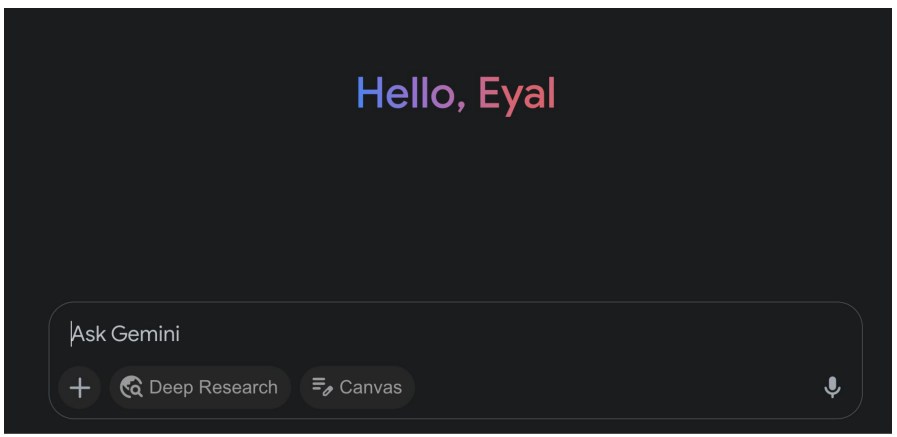
** More on GPT3 (and many other cool NLP models) later:)

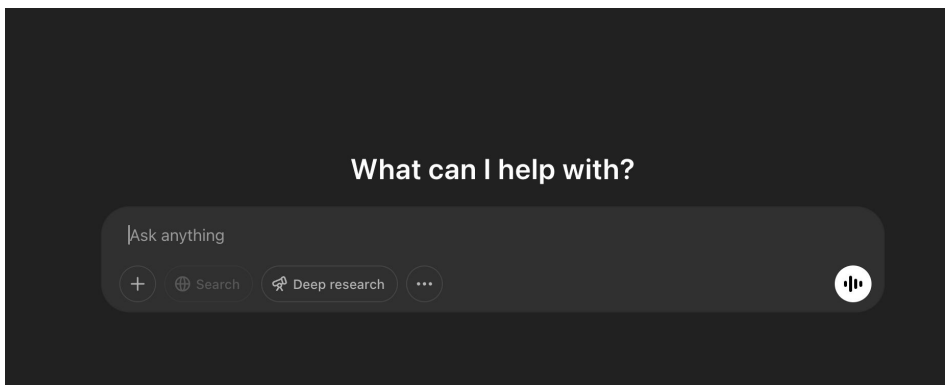
You use Language Models every day!





You use Language Models every day!





Language Modeling

Goal:

Assign a probability to a given sentence or sequence of words.

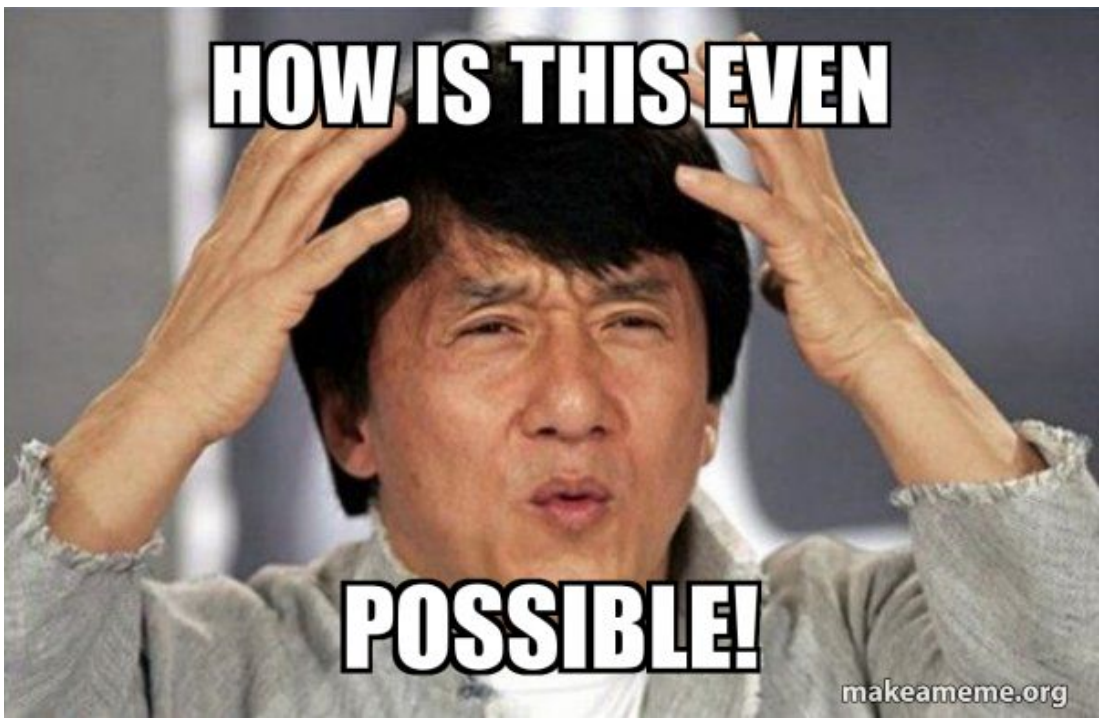
P(“Insanity is doing the same thing over and over again and expecting different results”) = ?

Another way of thinking about this:

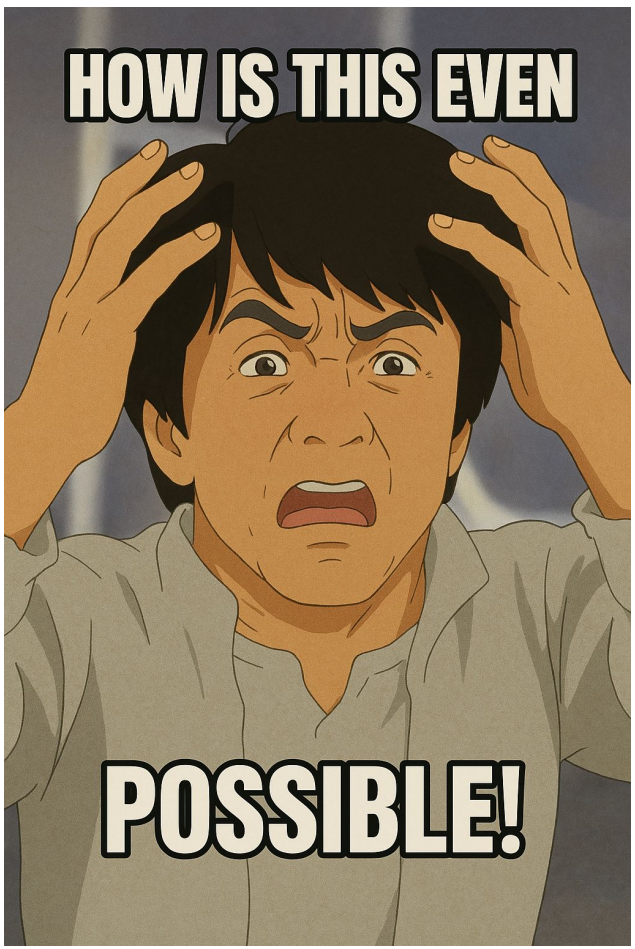
Given a sequence of words, what is likely to be the next word?

P(“expecting” | “Insanity is doing the same thing over and over again and… ”) = ?

How do we assign probability to text



How do we assign probability to text




EVERYTHING IS POSSIBLE
LET'S START

Probabilistic Language Model

Compute the probability of text

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

or

$$P(w_n|w_1, w_2, w_3, \ldots, w_{n-1})$$

and even

$$P(w_3|w_1, w_2, w_4, \ldots, w_n)$$

Probabilistic Language Model

P(As I foretold you, were all spirits, and
Are melted into air, into thin air:
And, like the baseless fabric of this vision,
The cloud-capp'd towers, the gorgeous palaces,
The solemn temples, the great globe itself,
Yea, all which it inherit, shall dissolve,
And, like this insubstantial pageant faded,
Leave not a rack behind. We are such stuff
As dreams are made on; and our little life
Is rounded with a sleep. Sir, I am vex'd;)

The Tempest Act IV, scene i, lines 148–158

Probabilistic Language Model

We define the joint probability of text W as follows:

$$\mathbf{P(W) = P(w_1, w_2, \ldots, w_n)}$$

Recall.. (Conditional distribution)

$P(B|A) = P(A,B)/P(A) \Rightarrow P(A,B) = P(B|A) \cdot P(A)$

Chain Rule for Text

We apply the chain rule to compute the joint probability of words in a sentence.

$$\mathbf{P(W) = P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdots P(w_n|w_1, w_2, \ldots, w_{n-1})}$$

Example:

P(“the dog barked”) = P(“the”)·P(“dog”|“the”)·P(“barked”|“the dog”)

The Maximum Likelihood Estimation

It turns out that the MLE of a word's conditional distribution is*:

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) = \frac{Counts(w_1, w_2, \ldots, w_n)}{Counts(w_1, w_2, \ldots, w_{n-1})}$$

Q: Is it realistic to estimate this?

* You may see the proof in the tutorial

The Maximum Likelihood Estimation

Given a training data, what does it takes to estimate the following:

$$P(night \mid the\ dog\ barked\ loudly\ last) = \frac{Counts(the\ dog\ barked\ loudly\ last\ nights)}{Counts(the\ dog\ barked\ loudly\ last)}$$

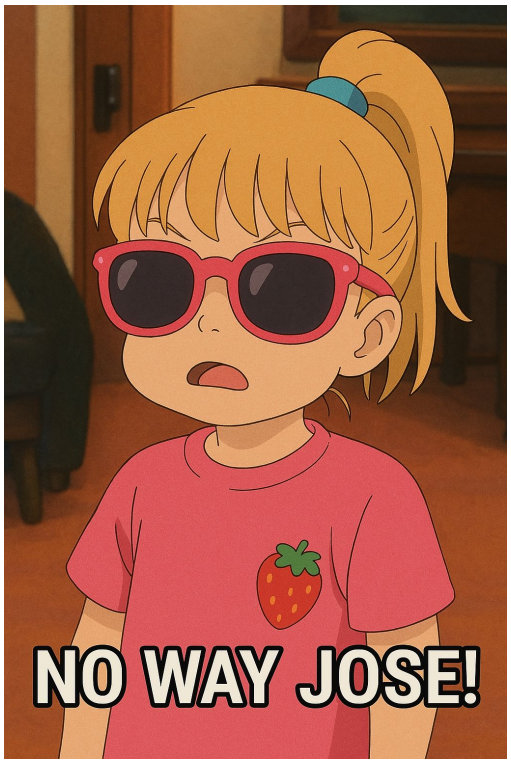
let's say

a. How many times would these expressions appear in text? 1/5 in the best case

b. Would that yield a reasonable estimate? No

can't do this - it's too sparce

We need some assumptions. Let’s use world knowledge.



We need some assumptions. Let’s use world knowledge.

Today's Agenda

- Language Modeling
- **N-gram Models**
- The Tagging Problem
- Hidden Markov Models

Markov Assumption

In general, a *k* order markov model assumes:

$$P(w_n|w_1, w_2,..., w_{n-1}) = P(w_n|w_{n-k},..., w_{n-1})$$

Back to our running example:

$$P(night \mid the\ dog\ barked\ loudly\ last) = P(night \mid last) = \frac{Counts(last\ nights)}{Counts(last)}$$

Bigram Language Model

Assumes a **first order markov** assumption.

A bigram language model consists of:

- **A (finite) vocabulary V**
- **A parameter p(w|u) for each possible bigram {u,w}**
  - **w ∈ V U {EOS}** end of sentence
  - **u ∈ V U {BOS}** beginning of sentence.

→ The Bigram MLE of text W is:

$$P(W) = \prod_{i=0}^{n} p(w_i \mid w_{i-1}) = \prod_{i=0}^{n} \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Bigram Language Model - Example

- **A (finite) vocabulary V**
- **A parameter P(w|u) for each possible bigram (u,w)**
  - **w ∈ V U {EOS}**
  - **u ∈ V U {BOS}**

$$P(the\ dog\ barked\ loudly\ last\ night) =$$

$$= P(the \mid BOS) * P(dog \mid the) * \ldots * P(night \mid last) * P(EOS \mid night)$$

Training a Bi-Gram Language Model

As always, we need to establish a model, a training procedure, and an inference procedure.

Given training data 'W', we want to find a model that maximizes P(W).

- **What are the model parameters?**
  - **We need to construct a *vocabulary* that represents the data.**
  - **We need to learn *p(w|u)* for any given bigram *{u,w}*.**

$$P\left(w|u\right) = \frac{count\left(u,w\right)}{count\left(u\right)}$$

Training - Example

Training data

The dog barked loudly last night.

The cat is afraid of the dog.

The mouse ate the cheese.

Training - Example

Training data

The dog barked loudly last night.

The cat is afraid of the dog.

The mouse ate the cheese.

Vocabulary
BOS
EOS
the
dog
barked
loudly
last
night
cat
is
afraid
of
mouse
ate
cheese

Training - Example

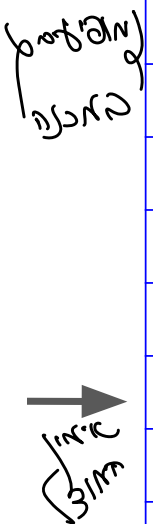
Training data

The dog barked loudly last night.

The cat is afraid of the dog.

The mouse ate the cheese.

Vocabulary
BOS
EOS
the
dog
barked
loudly
last
night
cat
is
afraid
of
mouse
ate
cheese



uni-gram counts		bi-gram counts	
BOS	3	BOS, the	3
EOS	3	the, dog	2
the	5	dog, barked	1
dog	2	barked, loudly	1
barked	1	loudly, last	1
loudly	1	last, night	1
last	1	night, EOS	1
night	1	the, cat	1
cat	1	cat, is	1
is	1	is, afraid	1
afraid	1	afraid, of	1
of	1	of, the	1
mouse	1	dog, EOS	1
ate	1	…	1
cheese	1	…	1

Inference

Well, this is quite simple:

$$P(W) = \prod_{i=0}^{n} p(w_i \mid w_{i-1}) = \prod_{i=0}^{n} \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

So, what's the probability of the following sentence?

The doctors recommended the patient not to eat rice.

Main Drawbacks of N-gram Language Model

- How do we decide what *N* would be?
  - The larger it is, what problems may occur?
  - The smaller it is, what are we missing?

- How to deal with OOV (unigrams, n-grams)?
  - A problem we many times refer as data sparseness

OOV - out of Vocabulary

Sparsity Problems with n-gram Language Models

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problems with n-gram Language Models

or OOV

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

→ 0 + count

Sparsity Problems with n-gram Language Models

$$\alpha_1 \text{count}(w_1) + \alpha_2 \text{count}(w_1 + w_2) + \alpha_3 \text{count}(w_1 + w_2 + w_3)$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

$$\max(\alpha_1, \alpha_2, \alpha_3) = \alpha_3$$

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

backoff

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse.* Typically, we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

Evaluation

- There is no ‘perfect’ way to evaluate the output of a LM
  - We generally see the same problems in text-generation

- The **standard** evaluation metric for Language Models is **perplexity**.

Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

Perplexity

- The standard **evaluation metric** for Language Models is **perplexity**.

$$PP(W) = \frac{1}{P(w_1 w_2 \ldots w_N)^{\frac{1}{N}}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

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

Chain Rule

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

Bi-gram assumption

Mid Overview

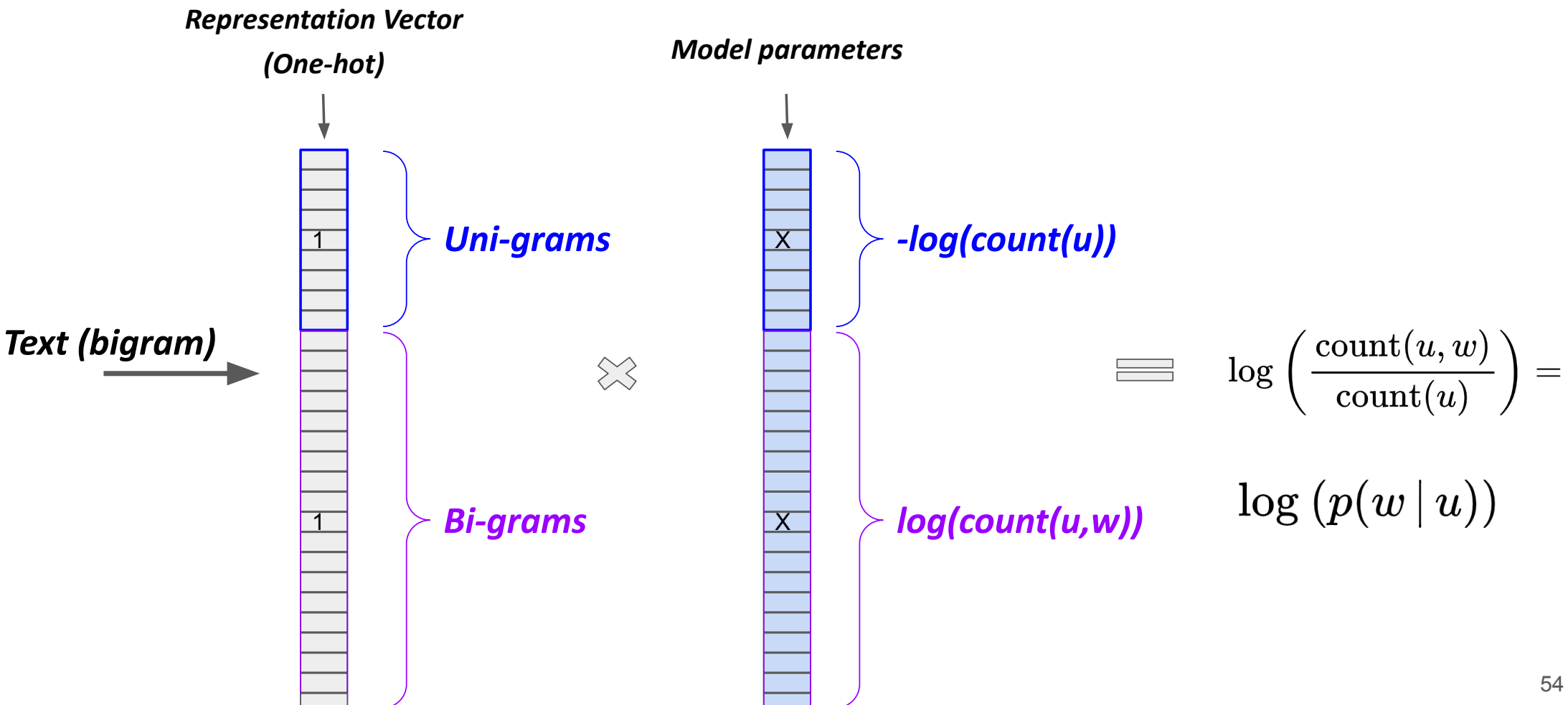
- Based on what we have seen so far, how do we represent text?

Mid Overview

- Based on what we have seen so far, how do we represent text?
  - Sparse vectors (uni-gram, bi-gram, etc..)
  - We can only represent (upto) N-grams

$$P(W) = \prod_{i=0}^{n} p(w_i \mid w_{i-1}) = \prod_{i=0}^{n} \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Bigram Model - a Vector Perspective



Mid Overview

- Based on what we have seen so far, how do we represent text?
  - Sparse vectors (uni-gram, bi-gram, etc..)
  - We can only represent (upto) N-grams

- Q: How would you improve representations?

Mid Overview

- Based on what we have seen so far, how do we represent text?
  - Sparse vectors (uni-gram, bi-gram, etc..)
  - We can only represent (upto) N-grams

- Q: How would you improve representations?
- A:
  - Represent more information (context, semantics, syntax)
  - Smaller (less sparse) representations

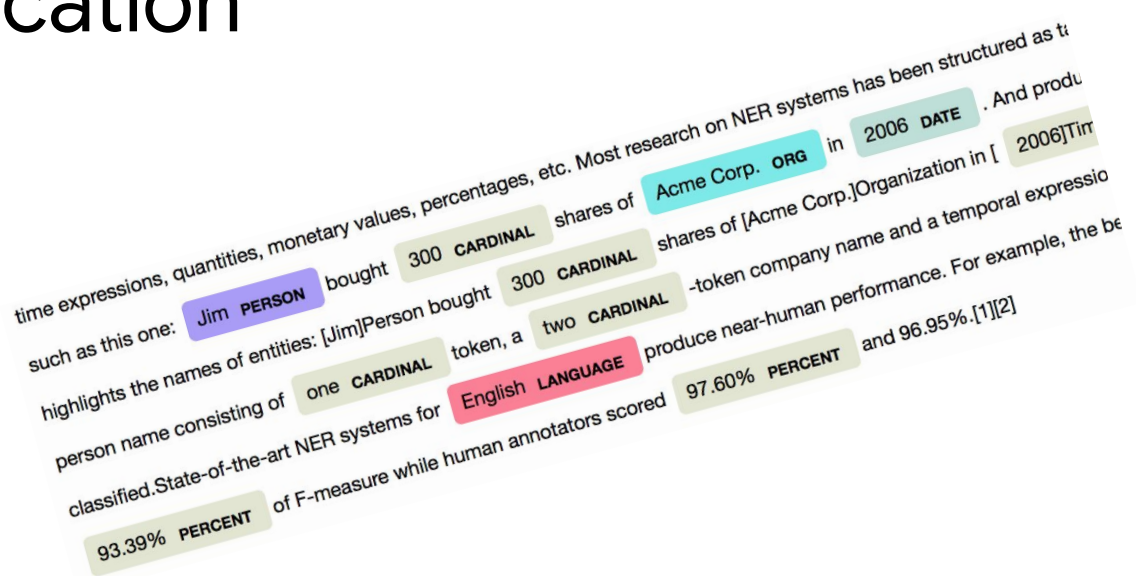
Today's Agenda

- Language Modeling
- N-gram Models
- **The Tagging Problem**
- Hidden Markov Models

(Recall ...) Token Classification

Input:

- Tokens/words $T=[t_1, t_2, \dots, t_k]$
- A set of classes $C = \{c_1, c_2, \dots, c_n\}$



Output:

- Predicted classes per token $\hat{C}=[\hat{c}_1, \hat{c}_2, \dots, \hat{c}_k]$, $\hat{c}_i \in C$ and $i \in \{1, \dots, k\}$

Part of Speech (POS) Tagging

Input:

The first congestion-free day that was held on Auguts 9, was a success.

Output:

Tags:

- DET - Determiner - ?
- ADJ - Adjective - תואר
- NN - Noun - שם עצם
- NUM - Number - מספר
- ...

VB - Verb - פועל

Part of Speech (POS) Tagging

So.. It looks easy, can we just map a word to its POS tag?

Part of Speech (POS) Tagging

So.. It looks easy, can we just map a word to POS tag?

- Ambiguity - there are words that may have several meaning.
  - was - once a verb and once an auxiliary

Consider the following example: "*The trash **can** is in the garage*"

- Local constrain -
  - the word can is more likely to be a modal-verb (MD) than a noun (NN)
- Contextual constraints -
  - **the can** - it is more likely to see a NN (than a MD) after a determiner (DT).


CAN WE USE
N-GRAM MODEL?


CAN WE USE
N-GRAM MODEL?

Today's Agenda

- Language Modeling
- N-gram Models
- The Tagging Problem
- **Hidden Markov Models**

Hidden Markov Models (HMM) classification

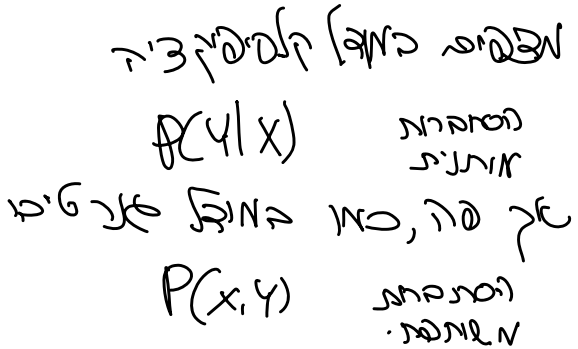
A generative model - defines p(x,y)

- Input: a sentence $X = x_1, x_2, \ldots, x_n$
- Output: sequence of tags $Y = y_1, y_2, \ldots, y_n$

HMM defines

$$\mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

for any sentence X and tags Y of the same length



How do we use HMMs for Token Classification?

How do we use HMMs for Token Classification?

Well, we will just find the argmax

$$\arg\max_{y_1, \ldots, y_n} \mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

Trigram HMM

Definition

For any sentence $x_1, x_2, ..., x_n$ and any sequence of tags $y_1, y_2, ..., y_{n+1}$, the joint probability of the sentence and the tags is:

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i \mid y_i)$$

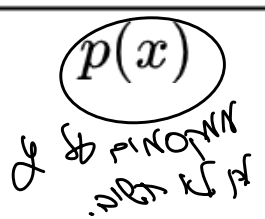
Assumptions:

- x_i = Vocab for i= 1, ..., n
- y_i = S (tag set) for i= 1, ..., n
- x_{-1}, x_0 = BOS
- y_{n+1} = EOS

How to Apply a Generative Model for Classification

$$p(X, Y) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i \mid y_i)$$

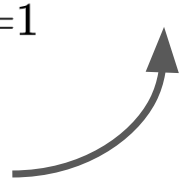
$$\arg\max_y p(y \mid x) = \arg\max_y \frac{p(x, y)}{p(x)} = \arg\max_y \frac{p(y) \cdot p(x \mid y)}{p(x)} = \arg\max_y p(y) \cdot p(x \mid y)$$



- HMM is a generative model:
  - q is the prior probability of the tags, i.e. p(y) (prior).
  - e is the conditional probabilities, i.e. p(x|y) (likelihood)

HMM - Name Origin

$$p(X, Y) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i \mid y_i)$$



Markov Chain of unobserved variables

HMM - Parameter Estimation (Training)

$$q(VB \mid DT,\ ADJ) = \frac{Count(DT,\ ADJ,\ VB)}{Count(ADJ,\ VB)}$$

$$e(was \mid VB) = \frac{Count(VB,\ was)}{Count(VB)}$$

Let's see an example

Assume the following corpus:

"the boy swims"

$\longrightarrow$

$$p(the,\ boy,\ swims, DT,\ NN,\ VB,\ EOS) =$$
$$q(DT \mid BOS,\ BOS) \cdot q(NN \mid BOS,\ DT) \cdot$$
$$q(VB \mid DT,\ NN) \cdot q(EOS \mid NN, VB)$$
$$e(the \mid DT) \cdot e(boy \mid NN) \cdot e(swims \mid VB)$$

Let's see an example

How do we perform inference?

$$\arg \max_{y_1, \ldots, y_n} \mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

We need to find the set of labels that is most appropriate for the new test example.


HOUSTON
WE HAVE A PROBLEM.

Searching the best set of labels

$$\arg \max_{y_1, \ldots, y_n} \mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

If we perform a brute force search, this might get messy:

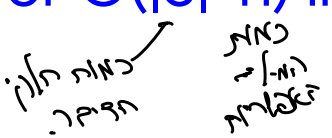
- Given an example of 5 words and 3 possible tags
  - How many combination of labels do we have?

Searching the best set of labels

$$\arg \max_{y_1, \ldots, y_n} \mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

If we perform a brute force search, this might get messy:

- Given an example of 5 words and 3 possible tags
  - How many combination of labels do we have? 3^5 or O(|S|^n) in the general case



Searching the best set of labels

$$\arg \max_{y_1, \ldots, y_n} \mathrm{p}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

- We can perform dynamic programming and reduce this search to O(N(|S|^3))

Let’s check it out!! :)

The Viterbi Algorithm

Key points:

$$p(X, Y) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i \mid y_i)$$

- Hidden states - Y
- Observations - X
- State transition probabilities

Dynamic programming - Use a matrix where each cell represents the probability of being in a state at a given time, considering the most probable path that led to that state

The Viterbi Algorithm

Dynamic programming - Use a matrix where each cell represents the probability of being in a state at a given time, considering the most probable path that led to that state

PI(3,u=DT,v=NNP) =

	NNP	DT	VB
NNP			
DT			
VB			

How do we calculate the probability of this state at this time?

"The Cat chased the mouse"

The Viterbi Algorithm

Dynamic programming - Use a matrix where each cell represents the probability of being in a state at a given time, considering the most probable path that led to that state

Noun Phrase

i=3

PI(3,u=DT,v=NNP) =

	NNP	DT	VB
NNP			
DT			
VB			

$$\max_{y_{i-2}} \prod_{i=1}^{3} q(NNP \mid y_{i-2}, DT) \cdot \prod_{i=1}^{3} e(\text{chased} \mid NNP)$$

0.2

i=2

	NNP	DT	VB
NNP		0.1	
DT		0.05	
VB		0.2	

y_{i-1}=DT

BOS = $y_{i-1} = y_{i-2}$: i=1 ב

How do we calculate the probability of this state at this time?

$$p(X, Y) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i \mid y_i)$$

The Viterbi Algorithm for HMMs

- Define n to be the length of the sentence
- Define S_k for $k = -1 \ldots n$ to be the set of possible tags at position k:

$$S_{-1} = S_0 = \{*\}$$

$$S_k = S \quad \text{for } k \in \{1 \ldots n\}$$

- Define

$$r(y_{-1}, y_0, y_1, \ldots, y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i)$$

- Define a dynamic programming table

$\pi(k, u, v)$ = maximum probability of a tag sequence ending in tags u, v at position k

that is,

$\pi(k, u, v) = \max_{\langle y_{-1}, y_0, y_1, \ldots, y_k \rangle : y_{k-1} = u, y_k = v} r(y_{-1}, y_0, y_1 \ldots y_k)$

A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

(Notation - *w* is a tag, x_k is the word in the kth position)

$$\text{Set } (y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n, u, v) \times q(\text{STOP}|u, v))$$

Notice how we're working backwards through the sentence, back to the base case (the beginning).

The Viterbi Algorithm with Backpointers

We want 'argmax', not 'max', i.e. the actual most-likely tag sequence.

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.

Initialization: Set $\pi(0,*,*) = 1$

Definition: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \dots n\}$

Algorithm:

- For $k = 1 \dots n$, *O(n)*
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$, *O(|S|·|S|)*

$$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)) \quad O(|S|)$$

$$bp(k,u,v) = \arg\max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

- Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
- For $k = (n-2) \dots 1$, $y_k = bp(k+2, y_{k+1}, y_{k+2})$
- **Return** the tag sequence $y_1 \dots y_n$

$\Rightarrow O(n|\mathcal{S}|^3)$ time in total

Which is much better than brute force search for <u>n>3</u>
$O(n|S|^3) << O(|S|^n)$

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

q(v|w)

	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Example - Viterbi

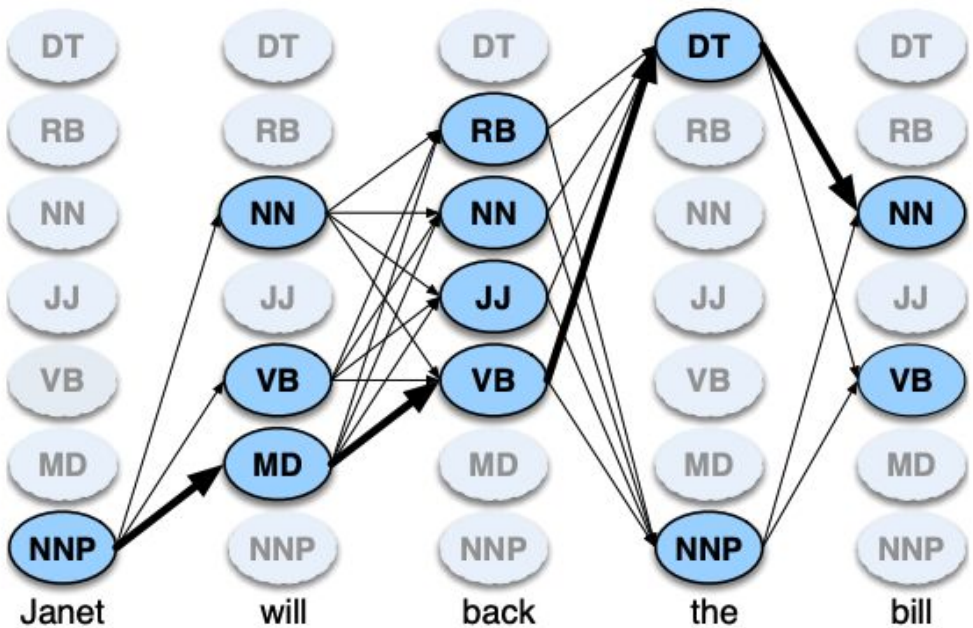
Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

e(x|v)

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*



	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

Initialization: Set $\pi(0, *, *) = 1$

PI(0,u,v)

	<BOS>	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	1	0	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0
MD	0	0	0	0	0	0	0	0
VB	0	0	0	0	0	0	0	0
JJ	0	0	0	0	0	0	0	0
NN	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0
DT	0	0	0	0	0	0	0	0

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

$$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

PI(1,u,v)

	<BOS>	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	0							
NNP								
MD								
VB								
JJ								
NN								
RB								
DT								

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

$$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

PI(1,u,v)

	<BOS>	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	0		0	0	0	0	0	0
NNP	0		0	0	0	0	0	0
MD	0		0	0	0	0	0	0
VB	0		0	0	0	0	0	0
JJ	0		0	0	0	0	0	0
NN	0		0	0	0	0	0	0
RB	0		0	0	0	0	0	0
DT	0		0	0	0	0	0	0

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

$$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

PI(1,u,v)

	<BOS>	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	0		0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0
MD	0	0	0	0	0	0	0	0
VB	0	0	0	0	0	0	0	0
JJ	0	0	0	0	0	0	0	0
NN	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0
DT	0	0	0	0	0	0	0	0

Example - Viterbi

Jant will back the bill → Janet/*NNP* will/*MD* back/*VB* the/*DT* bill/*NN*

$$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

PI(1,u=BOS,v=NNP)=
= 1*0.2767*0.000032

	<BOS>	NNP	MD	VB	JJ	NN	RB	DT
<BOS>	0	?	0	0	0	0	0	0
NNP	0	0	0	0	0	0	0	0
MD	0	0	0	0	0	0	0	0
VB	0	0	0	0	0	0	0	0
JJ	0	0	0	0	0	0	0	0
NN	0	0	0	0	0	0	0	0
RB	0	0	0	0	0	0	0	0
DT	0	0	0	0	0	0	0	0

Evaluation a POS model

There are two **standard** evaluation metric **Part of Speech tagging**

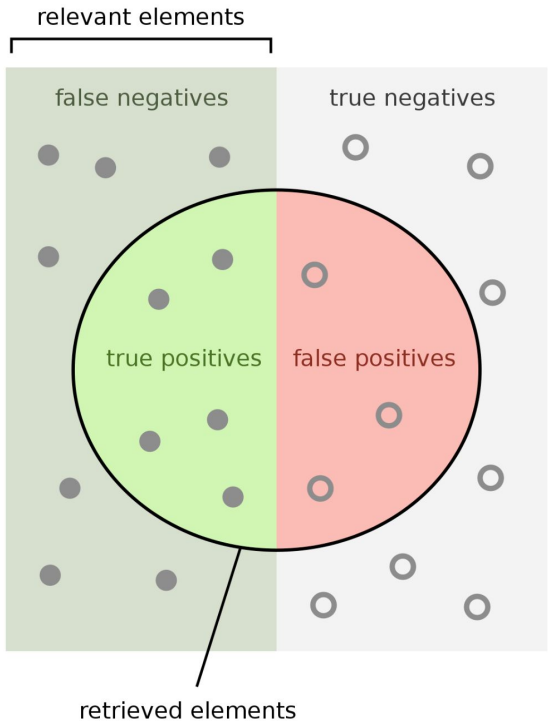
- Accuracy
  - Easy to understand and implement
  - Usually, when we have balanced labels

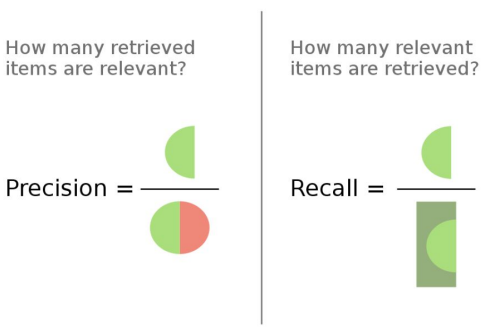
- F1
  - For unbalanced label space

The F1 Score

$$F_1 = \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}$$





Next week

- Discriminative view of language modeling
- Log linear models and feature engineering
- Log linear models for tagging
- Inference with dynamic programing