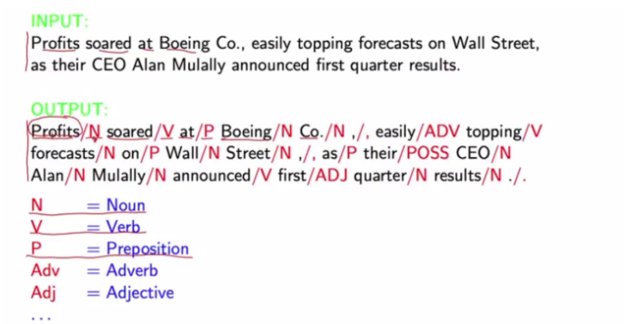
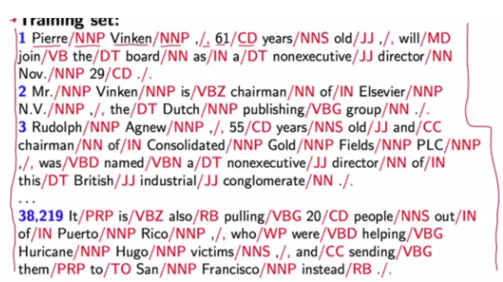
- **Part-of-Speech Tagging**
    - Input some sentence
    - Output have a tagged sequence each word is given an associated tag



- Named Entity Recognition
    - Input some sentence
    - Output named entities
        - A company, location, Person
- Named Entity Extraction as Tagging
    - Input some sentence
    - Output tagged sequence
        - NA = No entity
        - SC = Start company
        - CC = Continue Company
        - SL = Start location
        - CL = Continue Location
- Our Goal
    - Treat as machine learning problem
    - Training set
        - A set of sentences
        - Training sentences annotated by hand



    - From the training set, induce a function / algorithm that maps new sentences to their tag sequences
    - Treat this problem as a supervised learning problem
- Two types of Constraints
    - “Local”: e.g., can is more likely to be a model verb MD rather than a noun NN
        - has bias to be one part of speech over another part of speech
    - “Contextual”: e.g., a noun is much more likely than a verb to follow a determiner
    - Sometimes these preferences are in conflict

        - The trash can is in the garage
- **Generative models, and the noisy-channel model, for supervised learning**
- Supervised Learning Problems
    - We have training examples x^(i), y^(i) for i = 1…m. Each x^(i) is an input, each y^(i) is a label
    - Task is to learn a function f mapping inputs x to labels f(x)
        - x^(i) = the dog laughs, y^(i) DT NN VB
    - Conditional models:
        - Learn a distribution p(y|x) from training examples
            - will have various parameters
        - For any test input x, define f(x) = arg max_y p(y|x)
- Generative Models
    - We have training examples x^(i), y^(i) for i = 1…m. Task is to learn a function f mapping inputs x to labels f(x)
    - Generative models:
        - Learn a distribution p(x,y) from training examples
            - Joint distribution
        - Often we have p(x,y) = p(y)p(x|y)
            - Use bayes rule
            - p(y) prior likelihood
            - p(x|y) conditional generative model
                - the probability of generating x given y

▸ Note: we then have

$$\boxed{p(y|x)} = \frac{p(y)p(x|y)}{p(x)}$$

where $p(x) = \sum_y p(y)p(x|y)$

• p(y|x) DIRECTLY "DISCRIMINATIVE"

• p(x,y) GENERATIVE

        - given a joint distribution can derive a conditional distribution
- Decoding with Generative Models
    - We have training examples x^(i), y^(i) for i = 1…m. Task is to learn a function f mapping inputs x to labels f(x)
    - Generative models:
        - Learn a distribution p(x,y) from training examples
        - Often we have p(x,y) = p(y)p(x|y)

▸ Output from the model:

$$\begin{aligned} f(x) &= \arg\max_y p(y|x) \\ &= \arg\max_y \frac{p(y)p(x|y)}{p(x)} \\ &= \arg\max_y p(y)p(x|y) \end{aligned}$$

            - p(x) does not vary with y
- **Hidden Markov Model (HMM) taggers**
- Hidden Markov Models
    - have sentence x = x1 x2 x3 … xn

- We have a tag sequence y = y1 y2 y3 … yn
- We'll use an HMM to define

▸ We'll use an HMM to define

$$\boxed{p(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)}$$

for any sentence $x_1 \ldots x_n$ and tag sequence $y_1 \ldots y_n$ of the same length.

▸ Then the most likely tag sequence for x is

$$\arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1, y_2, \ldots, y_n)$$

  - p() defines a joint distribution over word sequences and tag sequences
  - the output is the tag sequence that maximizes the probability
    - the number of tag sequences grows exponential with n
      - therefore brute force is not going to work
- Trigram Hidden Markov Models (Trigram HMMs)

For any sentence $x_1 \ldots x_n$ where $x_i \in \mathcal{V}$ for $i = 1 \ldots n$, and any tag sequence $y_1 \ldots y_{n+1}$ where $y_i \in \mathcal{S}$ for $i = 1 \ldots n$, and $y_{n+1} = \boxed{\text{STOP}}$, the joint probability of the sentence and tag sequence is

$\mathcal{V} = \{$the, dog, cat, a, barks, $\}$

the dog barks   DT NN VB STOP

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

$S = \{$DT, NN, VB, P, ADV, …$\}$

where we have assumed that $x_0 = x_{-1} = *$.

e(the | DT)

~50 TAGS

Parameters of the model:

▸ $q(s|u, v)$ for any $s \in \mathcal{S} \cup \{\text{STOP}\}$, $u, v \in \mathcal{S} \cup \{*\}$ ⟸ TRIGRAM
▸ $e(x|s)$ for any $s \in \mathcal{S}$, $x \in \mathcal{V}$ ⟸ EMISSION PARAMETERS

  - Parameters
    - Trigram parameters
    - Conditional probability <- emission parameters e(xls)
- An example

If we have $n = 3$, $x_1 \ldots x_3$ equal to the sentence *the dog laughs*, and $y_1 \ldots y_4$ equal to the tag sequence $\boxed{\texttt{D N V STOP}}$, then

$$\begin{aligned} & p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) \\ = \; & q(\texttt{D}|*, *) \times q(\texttt{N}|*, \texttt{D}) \times q(\texttt{V}|\texttt{D}, \texttt{N}) \times q(\texttt{STOP}|\texttt{N}, \texttt{V}) \Leftarrow \\ & \times e(\textit{the}|\texttt{D}) \times e(\textit{dog}|\texttt{N}) \times e(\textit{laughs}|\texttt{V}) \end{aligned}$$

▸ STOP is a special tag that terminates the sequence
▸ We take $y_0 = y_{-1} = *$, where * is a special "padding" symbol

- Why the Name?
    - Hidden Markov Models

HIDDEN MARKOV MODELS

$$p(x_1 \ldots x_n, y_1 \ldots y_n) = \underbrace{\overbrace{q(\text{STOP}|y_{n-1}, y_n) \prod_{j=1}^{n} q(y_j | y_{j-2}, y_{j-1})}^{p(y_1, y_2 \ldots y_{n+1})}}_{\text{Markov Chain}} \times \underbrace{\prod_{j=1}^{n} e(x_j | y_j)}_{x_j\text{'s are observed}}$$

$p(x_1, \ldots x_n | y_1 \ldots y_{n+1})$

$p(x, y) = p(y) \times p(x|y)$

    - Prior probability over tag sequences
    - Second order Markov Chain
        - applied Markov to the problem of p(y)
    - p(x1…xnly1…yn+1)
        - probability of x conditioned on y
        - each word x is chosen only on the value of y
    - choose sequence of tags under this model
    - xj's are observed and yj's are unobserved
- **Hidden Markov Model (HMM) taggers Parameter estimation**
- Smoothed Estimation

$$q(\text{Vt} \mid \text{DT}, \text{JJ}) = \lambda_1 \times \frac{\text{Count(Dt, JJ, Vt)}}{\text{Count(Dt, JJ)}} + \lambda_2 \times \frac{\text{Count(JJ, Vt)}}{\text{Count(JJ)}} + \lambda_3 \times \frac{\text{Count(Vt)}}{\text{Count()}}$$

TRIG… ML ESTIMATE

BIGRAM ML ESTIMATE

UNIGRAM ML ESTIMATE

$\lambda_1 + \lambda_2 + \lambda_3 = 1$, and for all i, $\lambda_i \geq 0$

$$e(\text{base} \mid \text{Vt}) = \frac{\text{Count(Vt, base)}}{\text{Count(Vt)}}$$

$e(x|y) = 0$ for all y, if x is never seen in the training data

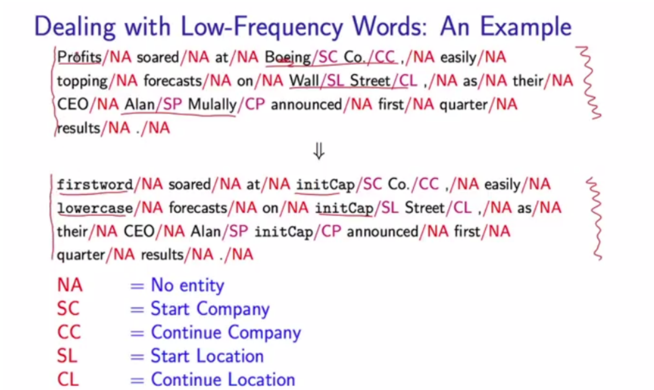
- Dealing with Low-Frequency Words: An Example
    - A common method is as follows:
        - Step1: Split vocabulary into two sets
            - Frequent words = words occurring >= 5 times in training
            - Low Frequency words = all other words
        - Step2: Map low frequency words into small, finite set, depending on prefixes, suffixes etc.

Dealing with Low-Frequency Words: An Example

[Bikel et. al 1999] **(named-entity recognition)**

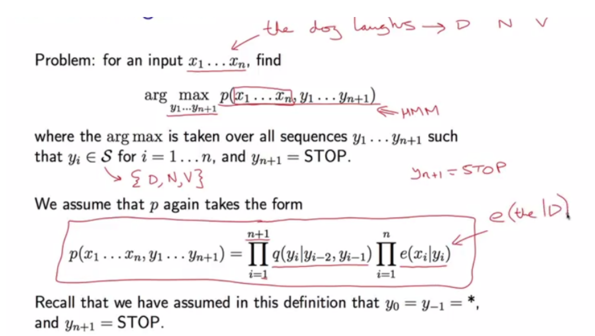
Word class	Example	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount, percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	first word of sentence	no useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other words

- lower frequency words mapped to pseudo words
    - first word
    - initCap
    - etc…

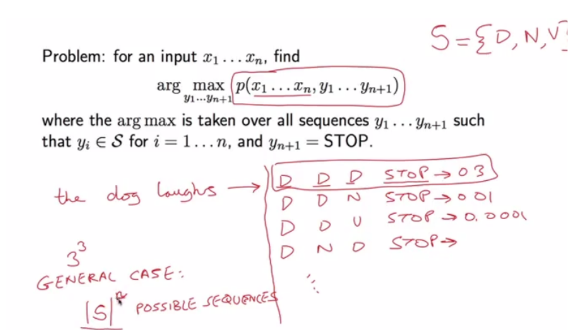


- build a Hidden Markov Model
    - e(firstword|na)
    - e(initCap|SC)
- closing the vocabulary
- mapping the low frequency words to a smaller method maintaining the spelling
- downside is it is heuristic

- **Hidden Markov Model The Viterbi algorithm**
- The Viterbi Algorithm



- Brute Force Search is Hopelessly Inefficient



- The Viterbi Algorithm
    - Define n to be the length of the input sentence

The Viterbi Algorithm

$x_1 \; x_2 \; x_3 \; \ldots \; x_n$

$* \; * \; D \; N \; V$

$y_{-1} \; y_0 \; y_1 \; y_2 \; y_3$

$k=3$

- 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

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

- r takes a sequence of tags as input
    - calculates its probability under the HMM under a truncated expression
- Define a dynamic programming table

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

k: $\{1, 2, \ldots n\}$; u: S_{k-1}; v: S_k

that is,

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

$y_{-1} = *$, $y_0 = *$

- An Example

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

$S = \{D, N, V, P\}$

$q(\;\;|\;\;)$

$e(\;\;|\;\;)$

*	*	D N V P	D N V P	D N V P	D N V P	D N V P	P	D
		The	man	saw	the	dog	with	the	telescope
		1	2	3	4	5	6	7	8

$\pi(7, P, D)$

        - pi() will be the maximum probability for any sequence that ends at position 6 and 7
        - Each sequence will have a probability by multiplying the q terms and e terms
- A Recursive Definition

Base case:

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

$* \; *$

Recursive definition:

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

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

k: $\{1, \ldots, n\}$; u: $\in S_{k-1}$; v: $\in S_k$

- Justification of the Recursive Definition

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

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

$S_5 = S = \{D,N,V,P\}$

$\boxed{\pi(6,N,P) \times q(D|N,P) \times e(the|D)}$

D N V P   P   D

The man saw the dog with the telescope
1 2 3 4 5 6 7 8

$$\pi(7,P,D) = \max_{w \in \{D,V,N,P\}} \left(\pi(6,w,P) \times q(D|w,P) \times e(the|D)\right)$$

- The Viterbi Algorithm

The Viterbi Algorithm

INPUT $= x_1 x_2 \dots x_n$
OUTPUT $= \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$

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 $k \in \{1 \dots n\}$ $\{D,N,V,P\}$

Algorithm:

- For $k = 1 \dots n$,   1, 2, 3, n
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,
    $$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right)$$
- **Return** $\max_{u \in \mathcal{S}_{n-1}, v \in \mathcal{S}_n} \left(\pi(n,u,v) \times q(\text{STOP}|u,v)\right)$

- The Viterbi Algorithm with Backpointers

The Viterbi Algorithm with Backpointers

INPUT $= x_1, \dots, x_n$
OUTPUT $= \arg\max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$

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 $k = 1 \dots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,
    $$\pi(k,u,v) = \max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right)$$
    $$bp(k,u,v) = \arg\max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right)$$
- Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} \left(\pi(n,u,v) \times q(\text{STOP}|u,v)\right)$
- For $k = (n-2) \dots 1$, $y_k = bp(k+2, y_{k+1}, y_{k+2})$
- **Return** the tag sequence $y_1 \dots y_n$

  - use dynamic programming to recover the arg max
  - runtime complexity is O(n|S|^3)
    - linear in the length of the sequence and brute force was exponential in time to the length of the sequence
- Pros and Cons
  - Hidden Markov Model taggers are very simple to train (just need to compile counts from the training corpus

- Perform relatively well
- Main difficulty is modeling
    - e(word|tag) can be very difficult if “words” are complex