CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Parts of Speech

• Perhaps starting with Aristotle in the West (384–322 BCE), there was the idea of having parts of speech
  • a.k.a lexical categories, word classes, “tags”, POS
• It comes from Dionysius Thrax of Alexandria (c. 100 BCE) the idea that is still with us that there are 8 parts of speech
  • But actually his 8 aren’t exactly the ones we are taught today
    • Thrax: noun, verb, article, adverb, preposition, conjunction, participle, pronoun
    • School grammar: noun, verb, adjective, adverb, preposition, conjunction, pronoun, interjection
Open vs. Closed classes

• Open vs. Closed classes
  • Closed:
    • determiners: *a, an, the*
    • pronouns: *she, he, I*
    • prepositions: *on, under, over, near, by, ...*
    • Why “closed”?
  • Open:
    • Nouns, Verbs, Adjectives, Adverbs.
POS Tagging

• Words often have more than one POS: *back*
  • The *back* door = JJ (adjective)
  • On my *back* = NN (noun)
  • Win the voters *back* = RB (adverb)
  • Promised to *back* the bill = VB (verb)

• The POS tagging problem is to determine the POS tag for a particular instance of a word.
POS Tagging

- **Input:** Plays well with others
- **Ambiguity:** NNS/VBZ UH/JJ/NN/RB IN NNS
- **Output:** Plays/VBZ well/RB with/IN others/NNS
- **Uses:**
  - Text-to-speech (how do we pronounce “lead”?)
  - Can write regexps like (Det) Adj* N+ over the output for phrases, etc.
  - As input to or to speed up a full parser
  - If you know the tag, you can back off to it in other tasks
POS tagging performance

• How many tags are correct? (Tag accuracy)
  • About 97% currently
  • But baseline is already 90%
    • Baseline is performance of stupidest possible method
      • Tag every word with its most frequent tag
      • Tag unknown words as nouns
  • Partly easy because
    • Many words are unambiguous
    • You get points for them (*the*, *a*, etc.) and for punctuation marks!
Deciding on the correct part of speech can be difficult even for people

- Mrs/NNP Shaefer/NNP never/RB got/VBD around/RP (particle) to/TO joining/VBG

- All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/IN (preposition) the/DT corner/NN

- Chateau/NNP Petrus/NNP costs/VCZ around/RB (adverb) 250/CD
How difficult is POS tagging?

- About 11% of the word types in the Brown corpus are ambiguous with regard to part of speech
- But they tend to be very common words. E.g., *that*
  - I know *that* he is honest = IN (preposition)
  - Yes, *that* play was nice = DT (determiner)
  - You can’t go *that* far = RB (adverb)
- 40% of the word tokens are ambiguous
Sources of information

• What are the main sources of information for POS tagging?
  • Knowledge of neighboring words
    • Bill saw that man yesterday
    • NNP NN DT NN NN
    • VB VB(D) IN VB NN
  • Knowledge of word probabilities
    • *man* is rarely used as a verb....

• The latter proves the most useful, but the former also helps
More and Better Features ➔
Feature-based tagger

• Can do surprisingly well just looking at a word by itself:
  • Word the: the → DT
  • Lowercased word Importantly: importantly → RB
  • Prefixes unfathomable: un- → JJ
  • Suffixes Importantly: -ly → RB
  • Capitalization Meridian: CAP → NNP
  • Word shapes 35-year: d-x → JJ
Hidden Markov Model
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.
Sample Markov Model for POS

- Det
- Noun
- PropNoun
- Verb
- stop

Transition probabilities:
- From start to Det: 0.1
- From start to Noun: 0.5
- From start to PropNoun: 0.4
- From start to Verb: 0.1
- From Det to Noun: 0.95
- From Noun to Verb: 0.9
- From PropNoun to Verb: 0.8
- From Verb to stop: 0.5
Sample Markov Model for POS

P(PropNoun Verb Det Noun) = 0.4*0.8*0.25*0.95*0.1=0.0076
Hidden Markov Model

• Probabilistic generative model for sequences.

• Assume an underlying set of \textit{hidden} (unobserved) states in which the model can be (e.g. part-of-speech).

• Assume probabilistic transitions between states over time (e.g. transition from POS to another POS as sequence is generated).

• Assume a \textit{probabilistic} generation of tokens from states (e.g. words generated for each POS).
Sample HMM for POS
Sample HMM Generation

Diagram:
- **Start**
  - 0.5 to PropNoun
  - 0.1 to Det
- **PropNoun**
  - 0.5 to PropNoun
  - 0.4 to Alice
  - 0.1 to John
- **Det**
  - 0.95 to Noun
  - 0.1 to the
- **Noun**
  - 0.9 to Verb
  - 0.1 to pen
- **Verb**
  - 0.5 to stop
  - 0.1 to hit
  - 0.25 to gave
  - 0.25 to played
  - 0.8 to bit
- **Stop**
Sample HMM Generation

The network diagram illustrates transitions between different parts of speech, such as determiners, nouns, and verbs, with associated probabilities. The diagram starts at the 'start' node, moves through determiners, nouns, and verbs, and ends at the 'stop' node. The transitions are labeled with probabilities, indicating the likelihood of moving from one category to another.
Sample HMM Generation

- **PropNoun**: John, Mary, Alice, Jerry
- **Det**: the, a, that
- **Noun**: cat, dog, bed, pen, apple
- **Verb**: bit, ate, saw, played, hit, gave
- **Transition Probabilities**:
  - Start to Det: 0.5
  - Det to PropNoun: 0.5, 0.1, 0.4
  - PropNoun to Noun: 0.25, 0.1
  - Noun to Verb: 0.1, 0.05, 0.9, 0.5, 0.1
  - Verb to Stop: 0.8, 0.25, 0.1

- **Emission Probabilities**:
  - Start to John: 0.1
  - John to PropNoun: 0.1
  - PropNoun to Noun: 0.95
  - Noun to Verb: 0.05, 0.1, 0.05, 0.1, 0.9
  - Verb to Stop: 0.8, 0.25, 0.1, 0.05, 0.1

- **Other**:
  - Tom
  - Bed
  - Apple
  - Stop
Sample HMM Generation

The diagram illustrates an HMM with states labeled as Determiner (Det), Propositional Noun (PropNoun), Noun, and Verb. The transitions between these states are shown with probabilities, such as 0.95 from Det to PropNoun, 0.9 from Noun to Verb, and so on. The emissions from each state are also indicated, such as "the", "a", "the", etc., from the Determiner state, and "cat", "dog", "bed", "pen", "apple", etc., from the Noun state. The start state is labeled as "start" with an arrow starting from it, and the stop state is labeled as "stop" with a loop returning to it. The probabilities on the transitions and emissions are shown as decimal values, such as 0.1, 0.5, etc.
Sample HMM Generation

```
John bit
```
Sample HMM Generation

John bit
John bit the
Sample HMM Generation

John bit the
Sample HMM Generation

John bit the apple
John bit the apple
Formally, Markov Sequences

- Consider a sequence of random variables $X_1, X_2, \ldots, X_m$ where $m$ is the length of the sequence.

- Each variable $X_i$ can take any value in $\{1, 2, \ldots, k\}$.

- How do we model the joint distribution

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$$
The Markov Assumption

\[
P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m) \\
= P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_1 = x_1, \ldots, X_{j-1} = x_{j-1}) \\
= P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_{j-1} = x_{j-1})
\]

- The first equality is exact (by the chain rule).
- The second equality follows from the Markov assumption: for all \( j = 2 \ldots m \),

\[
P(X_j = x_j | X_1 = x_1, \ldots, X_{j-1} = x_{j-1}) = P(X_j = x_j | X_{j-1} = x_{j-1})
\]
Homogeneous Markov Chains

- In a *homogeneous* Markov chain, we make an additional assumption, that for $j = 2 \ldots m$,

$$P(X_j = x_j | X_{j-1} = x_{j-1}) = q(x_j | x_{j-1})$$

where $q(x' | x)$ is some function

- Idea behind this assumption: the transition probabilities do not depend on the position in the Markov chain (do not depend on the index $j$)
Markov Models

Our model is then as follows:

\[ p(x_1, x_2, \ldots, x_m; \theta) = q(x_1) \prod_{j=2}^{m} q(x_j | x_{j-1}) \]

Parameters in the model:

- \( q(x) \) for \( x = \{1, 2, \ldots, k\} \)
  - Constraints: \( q(x) \geq 0 \) and \( \sum_{x=1}^{k} q(x) = 1 \)

- \( q(x'|x) \) for \( x = \{1, 2, \ldots, k\} \) and \( x' = \{1, 2, \ldots, k\} \)
  - Constraints: \( q(x'|x) \geq 0 \) and \( \sum_{x'=1}^{k} q(x'|x) = 1 \)
Probabilistic Models for Sequence Pairs

- We have two sequences of random variables: 
  \(X_1, X_2, \ldots, X_m\) and \(S_1, S_2, \ldots, S_m\)

- Intuitively, each \(X_i\) corresponds to an “observation” and each \(S_i\) corresponds to an underlying “state” that generated the observation. Assume that each \(S_i\) is in \(\{1, 2, \ldots, k\}\), and each \(X_i\) is in \(\{1, 2, \ldots, o\}\)

- How do we model the joint distribution

\[
P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)
\]
Probabilistic Models for Sequence Pairs

- We have two sequences of random variables: $X_1, X_2, \ldots, X_m$ and $S_1, S_2, \ldots, S_m$

  **Words** Part-of-Speech tags

- Intuitively, each $X_i$ corresponds to an “observation” and each $S_i$ corresponds to an underlying “state” that generated the observation. Assume that each $S_i$ is in $\{1, 2, \ldots, k\}$, and each $X_i$ is in $\{1, 2, \ldots, o\}$

- How do we model the joint distribution

  $P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)$
In HMMs, we assume that:

\[
P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)
= P(S_1 = s_1) \prod_{j=2}^{m} P(S_j = s_j | S_{j-1} = s_{j-1}) \prod_{j=1}^{m} P(X_j = x_j | S_j = s_j)
\]
Independence Assumptions in HMMs

- By the chain rule, the following equality is exact:

\[
P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m) = P(S_1 = s_1, \ldots, S_m = s_m) \times P(X_1 = x_1, \ldots, X_m = x_m | S_1 = s_1, \ldots, S_m = s_m)
\]

- Assumption 1: the state sequence forms a Markov chain

**e.g. Part-of-Speech tags**

\[
P(S_1 = s_1, \ldots, S_m = s_m) = P(S_1 = s_1) \prod_{j=2}^{m} P(S_j = s_j | S_{j-1} = s_{j-1})
\]
By the chain rule, the following equality is exact:

\[
P(X_1 = x_1, \ldots, X_m = x_m | S_1 = s_1, \ldots, S_m = s_m) = \prod_{j=1}^{m} P(X_j = x_j | S_1 = s_1, \ldots, S_m = s_m, X_1 = x_1, \ldots X_{j-1} = x_j)
\]

Assumption 2: each observation depends only on the underlying state

\[
P(X_j = x_j | S_1 = s_1, \ldots, S_m = s_m, X_1 = x_1, \ldots X_{j-1} = x_j) = P(X_j = x_j | S_j = s_j)
\]
Formally

- The model takes the following form:

\[
p(x_1 \ldots x_m, s_1 \ldots s_m; \theta) = t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j)
\]

- Parameters in the model:

1. Initial state parameters \( t(s) \) for \( s \in \{1, 2, \ldots, k\} \)
2. Transition parameters \( t(s'|s) \) for \( s, s' \in \{1, 2, \ldots, k\} \)
3. Emission parameters \( e(x|s) \) for \( s \in \{1, 2, \ldots, k\} \) and \( x \in \{1, 2, \ldots, o\} \)
HMM

• Parameter estimation
  • Learning the probabilities from training data
    • P(verb|noun)?, P(apple|noun)?

• Inference: Viterbi algorithm (dynamic programming)
  • Given a new sentence, what are the POS tags for the words?
HMM

• Parameter estimation

• Inference: Viterbi algorithm (dynamic programming)
We’ll now discuss parameter estimates in the case of fully observed data: for $i = 1 \ldots n$, we have pairs of sequences $x_{i,j}$ for $j = 1 \ldots m$ and $s_{i,j}$ for $j = 1 \ldots m$. (i.e., we have $n$ training examples, each of length $m$.)
Parameter Estimation: Transition Parameters

• $P(\text{verb}|\text{noun})$?
Assume we have fully observed data: for \( i = 1 \ldots n \), we have pairs of sequences \( x_{i,j} \) for \( j = 1 \ldots m \) and \( s_{i,j} \) for \( j = 1 \ldots m \).

Define \( \text{count}(i, s \rightarrow s') \) to be the number of times state \( s' \) follows state \( s \) in the \( i \)'th training example. More formally:

\[
\text{count}(i, s \rightarrow s') = \sum_{j=1}^{m-1} \left[ [s_{i,j} = s \land s_{i,j+1} = s'] \right]
\]

(We define \([\pi]\) to be 1 if \( \pi \) is true, 0 otherwise.)

The maximum-likelihood estimates of transition probabilities are then

\[
t(s' | s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \rightarrow s')}{\sum_{i=1}^{n} \sum_{s'} \text{count}(i, s \rightarrow s')}
\]
Parameter Estimation: Emission Parameters

• \( P(\text{apple} | \text{noun}) \)?
Assume we have fully observed data: for $i = 1 \ldots n$, we have pairs of sequences $x_{i,j}$ for $j = 1 \ldots m$ and $s_{i,j}$ for $j = 1 \ldots m$.

Define $\text{count}(i, s \leadsto x)$ to be the number of times state $s$ is paired with emission $x$. More formally:

$$\text{count}(i, s \leadsto x) = \sum_{j=1}^{m} [[s_{i,j} = s \land x_{i,j} = x]]$$

The maximum-likelihood estimates of emission probabilities are then

$$e(x|s) = \frac{\sum_{i=1}^{n} \text{count}(i, s \leadsto x)}{\sum_{i=1}^{n} \sum_{x} \text{count}(i, s \leadsto x)}$$
Parameter Estimation: Initial State Parameters

- Assume we have fully observed data: for $i = 1 \ldots n$, we have pairs of sequences $x_{i,j}$ for $j = 1 \ldots m$ and $s_{i,j}$ for $j = 1 \ldots m$.

- Define $\text{count}(i, s)$ to be 1 if state $s$ is the initial state in the sequence, and 0 otherwise:

$$\text{count}(i, s) = [s_{i,1} = s]$$

- The maximum-likelihood estimates of initial state probabilities are:

$$t(s) = \frac{\sum_{i=1}^{n} \text{count}(i, s)}{n}$$
HMM

- Parameter estimation

- Inference: Viterbi algorithm (dynamic programming)
The Viterbi Algorithm

- Goal: for a given input sequence $x_1, \ldots, x_m$, find

  $$\arg \max_{s_1, \ldots, s_m} p(x_1 \ldots x_m, s_1 \ldots s_m; \theta)$$

- This is the most likely state sequence $s_1 \ldots s_m$ for the given input sequence $x_1 \ldots x_m$
• Continue forward in time until reaching final time point.
• **The goal:** find a path with highest probability
The Viterbi Algorithm

- Goal: for a given input sequence $x_1, \ldots, x_m$, find

$$\arg \max_{s_1, \ldots, s_m} p(x_1 \ldots x_m, s_1 \ldots s_m; \theta)$$

- The Viterbi algorithm is a dynamic programming algorithm. Basic data structure:

$$\pi[j, s]$$

will be a table entry that stores the maximum probability for any state sequence ending in state $s$ at position $j$. More formally: $\pi[1, s] = t(s)e(x_1|s)$, and for $j > 1$,

$$\pi[j, s] = \max_{s_1 \ldots s_{j-1}} \left[ t(s_1)e(x_1|s_1) \left( \prod_{k=2}^{j-1} t(s_k|s_{k-1})e(x_k|s_k) \right) t(s|s_{j-1})e(x_j|s) \right]$$
The Viterbi Algorithm

- Initialization: for $s = 1 \ldots k$
  \[ \pi[1, s] = t(s)e(x_1|s) \]

- For $j = 2 \ldots m$, $s = 1 \ldots k$:
  \[ \pi[j, s] = \max_{s' \in \{1 \ldots k\}} [\pi[j - 1, s'] \times t(s|s') \times e(x_j|s)] \]

- We then have
  \[ \max_{s_1 \ldots s_m} p(x_1 \ldots x_m, s_1 \ldots s_m; \theta) = \max_s \pi[m, s] \]

- The algorithm runs in $O(mk^2)$ time
Viterbi Backpointers
Viterbi Backtrace

Most likely Sequence: $s_0 \, s_N \, s_1 \, s_2 \, \ldots \, s_2 \, s_F$
The Viterbi Algorithm: Backpointers

- Initialization: for \( s = 1 \ldots k \)
  \[
  \pi[1, s] = t(s)e(x_1|s)
  \]

- For \( j = 2 \ldots m, s = 1 \ldots k \):
  \[
  \pi[j, s] = \max_{s' \in \{1 \ldots k\}} \left[ \pi[j - 1, s'] \times t(s|s') \times e(x_j|s) \right]
  \]
  and
  \[
  bp[j, s] = \arg \max_{s' \in \{1 \ldots k\}} \left[ \pi[j - 1, s'] \times t(s|s') \times e(x_j|s) \right]
  \]

- The \( bp \) entries are backpointers that will allow us to recover the identity of the highest probability state sequence.
Highest probability for any sequence of states is

$$\max_s \pi[m, s]$$

To recover identity of highest-probability sequence:

$$s_m = \arg \max_s \pi[m, s]$$

and for \( j = m \ldots 2, \)

$$s_{j-1} = bp[j, s_j]$$

The sequence of states \( s_1 \ldots s_m \) is then

$$\arg \max_{s_1, \ldots, s_m} p(x_1 \ldots x_m, s_1 \ldots s_m; \theta)$$
What We Learned Today

• Text Categorization
• Naïve Bayes
• Part-of-speech tagging
• Hidden Markov Models
Homework

• Reading J&M ch5&6

• Assignment 1 is out. Due Oct 9.

• Start thinking about course project and find a team
  • Project proposal due Oct 2