Lecture 7: Sequence Labeling

Recap: Statistical POS tagging with HMMs

She promised to back the bill

\[ w = w^{(1)} w^{(2)} w^{(3)} w^{(4)} w^{(5)} w^{(6)} \]

\[ t = t^{(1)} t^{(2)} t^{(3)} t^{(4)} t^{(5)} t^{(6)} \]

PRP VBD TO VB DT NN

What is the most likely sequence of tags \( t = t^{(1)} \ldots t^{(N)} \) for the given sequence of words \( w = w^{(1)} \ldots w^{(N)} \)?

\[ t^* = \arg\max_t P(t \mid w) \]

Recap: Statistical POS tagging

POS tagging with generative models

\[
\arg\max_t P(t \mid w) = \arg\max_t \frac{P(t, w)}{P(w)} = \arg\max_t P(t, w) = \arg\max_t P(t) P(w \mid t)
\]

\( P(t, w) \): the joint distribution of the labels we want to predict (\( t \)) and the observed data (\( w \)).
We decompose \( P(t, w) \) into \( P(t) \) and \( P(w \mid t) \) since these distributions are easier to estimate.

Models based on joint distributions of labels and observed data are called generative models: think of \( P(t) P(w \mid t) \) as a stochastic process that first generates the labels, and then generates the data we see, based on these labels.
Hidden Markov Models (HMMs)

HMMs are generative models for POS tagging (and other tasks, e.g. in speech recognition).

**Independence assumptions of HMMs**

\( P(t) \) is an \( n \)-gram model over tags:

- Bigram HMM: \( P(t) = P(t_1)P(t_2 | t_1)P(t_3 | t_2) \ldots P(t_N | t_{N-1}) \)
- Trigram HMM: \( P(t) = P(t_1)P(t_2 | t_1)P(t_3 | t_2, t_1) \ldots P(t_N | t_{N-1}, t_{N-2}) \)

\( P(t_i) \) or \( P(t_i | t_j, t_k) \) are called transition probabilities.

In \( P(w | t) \) each word is generated by its own tag:

\( P(w) = P(w_1 | t_1)P(w_2 | t_2) \ldots P(w_N | t_N) \)

\( P(w | t) \) are called emission probabilities.

Viterbi algorithm

**Task:** Given an HMM, return most likely tag sequence \( t^{(1)} \ldots t^{(N)} \) for a given word sequence (sentence) \( w^{(1)} \ldots w^{(N)} \)

**Data structure (Trellis):** \( N \times T \) table for sentence \( w^{(1)} \ldots w^{(N)} \) and tag set \( \{t_1, \ldots, t_T\} \). Cell \( \text{trellis}[i][j] \) stores score of best tag sequence for \( w^{(1)} \ldots w^{(j)} \) that ends in tag \( t_j \) and a backpointer to the cell corresponding to the tag of the preceding word \( \text{trellis}[i-1][k] \)

**Basic procedure:**
- Fill trellis from left to right
- Initialize \( \text{trellis}[1][k] := P(t_k) \times P(w^{(1)} | t_k) \)
- For \( \text{trellis}[i][j] \):
  - Find best preceding tag \( k^* = \arg \max_k (\text{trellis}[i-1][k] \times P(t_j | t_k)) \)
  - Add backpointer from \( \text{trellis}[i][j] \) to \( \text{trellis}[i-1][k^*] \)
  - Set \( \text{trellis}[i][j] := \text{trellis}[i-1][k^*] \times P(t_j | t_{k^*}) \times P(w^{(i)} | t_j) \)
- Return tag sequence that ends in the highest scoring cell \( \arg \max_k (\text{trellis}[N][k]) \) in the last column

Other HMM algorithms

**The Forward algorithm:**
Computes \( P(w) \) by replacing Viterbi’s `max()` with `sum`

**Learning HMMs from raw text with the EM algorithm:**
- We have to replace the observed counts (from labeled data) with expected counts (according to the current model)
- Renormalizing these expected counts will give a new model
- This will be “better” than the previous model, but we will have to repeat this multiple times to get to decent model

**The Forward-Backward algorithm:**
A dynamic programming algorithm for computing the expected counts of tag bigrams and word-tag occurrences in a sentence under a given HMM
**Sequence labeling**

**Task:** assign POS tags to words

**POS tagging**

Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

Pierre_NNP Vinken_NNP , , 61_CD years_NNS old JJ , , will_MD join_VB IBM_NNP ‘s_POS board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD . .

**Task:** identify all non-recursive NP chunks

**Noun phrase (NP) chunking**

Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.


**The BIO encoding**

We define three new tags:
- **B-NP:** beginning of a noun phrase chunk
- **I-NP:** inside of a noun phrase chunk
- **O:** outside of a noun phrase chunk


Pierre_B-NP Vinken_I-NP , O 61_B-NP years_I-NP old_O , O will_0 join_O IBM_B-NP ‘s_O board_B-NP as_O a_B-NP nonexecutive_I-NP director_I-NP Nov._B-NP 29_I-NP . .
Shallow parsing

Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

[NP Pierre Vinken], [NP 61 years] old, [VP will join] [NP IBM] ‘s [NP board] [PP as] [NP a nonexecutive director] [NP Nov. 2].

Task: identify all non-recursive NP, verb (“VP”) and preposition (“PP”) chunks

The BIO encoding for shallow parsing

We define several new tags:
- B-NP B-VP B-PP: beginning of an NP, “VP”, “PP” chunk
- I-NP I-VP I-PP: inside of an NP, “VP”, “PP” chunk
- O: outside of any chunk

[NP Pierre Vinken], [NP 61 years] old, [VP will join] [NP IBM] ‘s [NP board] [PP as] [NP a nonexecutive director] [NP Nov. 2].

Pierre B-NP Vinken I-NP _O 61 B-NP years I-NP old O _O will B-VP join I-VP IBM B-NP ‘s O board B-NP as B-PP a B-NP nonexecutive I-NP director I-NP Nov. B-NP 29 I-NP _O

Named Entity Recognition

Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

[PERS Pierre Vinken], 61 years old, will join [ORG IBM] ‘s board as a nonexecutive director [DATE Nov. 2].

Task: identify all mentions of named entities (people, organizations, locations, dates)

The BIO encoding for NER

We define many new tags:
- B-PERS, B-DATE, …: beginning of a mention of a person/date...
- I-PERS, I-DATE, …: inside of a mention of a person/date...
- O: outside of any mention of a named entity

[PERS Pierre Vinken], 61 years old, will join [ORG IBM] ‘s board as a nonexecutive director [DATE Nov. 2].

Pierre B-PERS Vinken I-PERS _O 61_0 years_0 old_0 _O will_0 join_0 IBM B-ORG ‘s_0 board_0 as_0 a_0 nonexecutive_0 director_0 Nov._B-DATE 29 I-DATE _O
Many NLP tasks are sequence labeling tasks

**Input:** a sequence of tokens/words:

Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

**Output:** a sequence of labeled tokens/words:

**POS-tagging:** Pierre _NNP_ Vinken _NNP_ , __, 61 _CD_ years _NNS_ old _JJ_ , __, will _MD_ join _VB_ IBM _NNP_ ‘s _POS_ board _NN_ as _IN_ a _DT_ nonexecutive _JJ_ director _NN_ Nov. _NNP_ 29 _CD_ __.

**Named Entity Recognition:** Pierre _B-PERS_ Vinken _I-PERS_ , _O_ 61 _O_ years _O_ old _O_ , _O_ will _O_ join _O_ IBM _B-ORG_ ‘s _O_ board _O_ as _O_ a _O_ nonexecutive _O_ director _O_ Nov. _B-DATE_ 29 _I-DATE_ _O_.

Directed graphical models

Graphical models are a notation for probability models. In a directed graphical model, each node represents a distribution over a random variable:

- \( P(X) = \boxed{\begin{bmatrix} X \end{bmatrix}} \)

**Arrows** represent dependencies (they define what other random variables the current node is conditioned on)

- \( P(Y) P(X | Y) = \begin{bmatrix} Y \rightarrow X \end{bmatrix} \)

- \( P(Y) P(Z) P(X | Y, Z) = \begin{bmatrix} Y \rightarrow X \rightarrow Z \end{bmatrix} \)

**Shaded nodes** represent observed variables. **White nodes** represent hidden variables

- \( P(Y) P(X | Y) \) with Y hidden and X observed = \( \boxed{\begin{bmatrix} Y \rightarrow X \end{bmatrix}} \)

HMMs as graphical models

HMMs are generative models of the observed input string \( \mathbf{w} \)

They ‘generate’ \( \mathbf{w} \) with \( P(\mathbf{w}, \mathbf{t}) = \prod_{i} P(t^{(i)} | t^{(i-1)})P(w^{(i)} | t^{(i)}) \)

When we use an HMM to tag, we observe \( \mathbf{w} \), and need to find \( \mathbf{t} \)

\[ \begin{array}{cccc}
\mathbf{t}^{(1)} & \mathbf{t}^{(2)} & \mathbf{t}^{(3)} & \mathbf{t}^{(4)} \\
\mathbf{W}^{(1)} & \mathbf{W}^{(2)} & \mathbf{W}^{(3)} & \mathbf{W}^{(4)} \\
\end{array} \]
Models for sequence labeling

**Sequence labeling:** Given an input sequence $w = w^{(1)} \ldots w^{(n)}$, predict the best (most likely) label sequence $t = t^{(1)} \ldots t^{(n)}$

$$\arg\max_t P(t|w)$$

**Generative models** use Bayes Rule:

$$\arg\max_t P(t|w) = \arg\max_t \frac{P(t, w)}{P(w)} = \arg\max_t P(t, w) = \arg\max_t P(t)P(w|t)$$

**Discriminative (conditional) models** model $P(t|w)$ directly

Advantages of discriminative models

**We’re usually not really interested in** $P(w|t)$.
- $w$ is given. We don’t need to predict it!
Why not model what we’re actually interested in: $P(t|w)$

**Modeling $P(w|t)$ well is quite difficult:**
- Prefixes (capital letters) or suffixes are good predictors for certain classes of $t$ (proper nouns, adverbs, …)
- So we don’t want to model words as atomic symbols, but in terms of features
- These features may also help us deal with unknown words
- But features may not be independent

**Modeling $P(t|w)$ with features should be easier:**
- Now we can incorporate arbitrary features of the word, because we don’t need to predict $w$ anymore

**Discriminative probability models**

A discriminative or **conditional** model of the labels $t$ given the observed input string $w$ models $P(t|w) = \prod_i P(t^{(i)}|w^{(i)}, t^{(i-1)})$ directly.

Discriminative models

There are two main types of discriminative probability models:
- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

**MEMMs and CRFs:**
- are both based on logistic regression
- have the same graphical model
- require the Viterbi algorithm for tagging
- differ in that MEMMs consist of independently learned distributions, while CRFs are trained to maximize the probability of the entire sequence
Probabilistic classification

Classification:
Predict a class (label) $c$ for an input $x$
There are only a (small) finite number of possible class labels

Probabilistic classification:
- Model the probability $P( c \mid x)$
  $P(c\mid x)$ is a probability if $0 \leq P( c_i \mid x) \leq 1$, and $\sum_i P( c_i \mid x) = 1$
- Return the class $c^* = \arg\max_i P( c_i \mid x)$
  that has the highest probability

One standard way to model $P( c \mid x)$ is logistic regression (used by MEMMs and CRFs)

Using features

Think of feature functions as useful questions you can ask about the input $x$:

- Binary feature functions:
  $f_{\text{first-letter-capitalized}}(\text{Urbana}) = 1$
  $f_{\text{first-letter-capitalized}}(\text{computer}) = 0$

- Integer (or real-valued) features:
  $f_{\text{number-of-vowels}}(\text{Urbana}) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

From features to probabilities

We associate a real-valued weight $w_{ic}$ with each feature function $f_i(x)$ and output class $c$
Note that the feature function $f_i(x)$ does not have to depend on $c$ as long as the weight does (note the double index $w_{ic}$)
This gives us a real-valued score for predicting class $c$ for input $x$: $\text{score}(x,c) = \sum_i w_{ic} f_i(x)$

This score could be negative, so we exponentiate it: $\text{score}(x,c) = \exp( \sum_i w_{ic} f_i(x))$
To get a probability distribution over all classes $c$, we renormalize these scores:
$P(c \mid x) = \frac{\text{score}(x,c)}{\sum_i \text{score}(x,c_i)}$
$= \frac{\exp( \sum_i w_{ic} f_i(x))}{\sum_j \exp( \sum_i w_{ij} f_i(x))}$

Learning: finding $w$

Learning = finding weights $w$
We use conditional maximum likelihood estimation (and standard convex optimization algorithms) to find/learn $w$
(for more details, attend CS446 and CS546)

The conditional MLE training objective:
Find the $w$ that assigns highest probability to all observed outputs $c_i$ given the inputs $x_i$

$\hat{w} = \arg\max_w \prod_i P(c_i \mid x_i, w)$
Terminology

Models that are of the form
\[
P(c \mid x) = \frac{\text{score}(x, c)}{\sum_j \text{score}(x, c_j)} = \exp\left(\sum_{i} w_{ic} f_i(x)\right) / \sum_j \exp\left(\sum_{i} w_{ij} f_i(x)\right)
\]
are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models.

CS446 and CS546 should give you more details about these.

The normalizing term \(\sum_j \exp\left(\sum_{i} w_{ij} f_i(x)\right)\) is also called the partition function and is often abbreviated as \(Z\).

Maximum Entropy Markov Models

MEMMs use a MaxEnt classifier for each \(P(t_i(i) \mid w(i), t(i-1))\):

\[
P(t_i(i) = t_k \mid t(i-1), w(i)) = \frac{\exp\left(\sum_{j} \lambda_{jk} f_j(t(i-1), w(i))\right)}{\sum_l \exp\left(\sum_{j} \lambda_{jl} f_j(t(i-1), w(i))\right)}
\]

Since we use \(w\) to refer to words, let’s use \(\lambda_{jk}\) as the weight for the feature function \(f_j(t(i-1), w(i))\) when predicting tag \(t_k\):

Viterbi for MEMMs

\(\text{trellis}[n][i]\) stores the probability of the most likely (Viterbi) tag sequence \(t(1)\ldots(n)\) that ends in tag \(t_i\) for the prefix \(w(1)\ldots w(n)\).

Remember that we do not generate \(w\) in MEMMs. So:
\[
\text{trellis}[n][i] = \max_{t(1)\ldots(n-1)} [ P(t(1)\ldots(n-1), t(n) = t_i \mid w(1)\ldots(n)) ] = \max_j [ \text{trellis}[n-1][j] \times P(t_i(t), w(n))] = \max_j [ \max_{t(1)\ldots(n-2)} [P(t(1)\ldots(n-2), t(n-1) = t_j \mid w(1)\ldots(n-1))] \times P(t_i(t), w(n))]\]

Today’s key concepts

Sequence labeling tasks:
- POS tagging
- NP chunking
- Shallow Parsing
- Named Entity Recognition

Discriminative models:
- Maximum Entropy classifiers
- MEMMs
Supplementary material: Other HMM algorithms (very briefly...)

Learning an HMM from unlabeled text

We can't count anymore. We have to guess how often we'd expect to see \( t_i \) etc. in our data set.

Call this expected count \( \langle C(... \rangle \)

- Our estimate for the transition probabilities:
  \[
  \hat{P}(t_j | t_i) = \frac{\langle C(t_i, t_j) \rangle}{\langle C(t_i) \rangle}
  \]

- Our estimate for the emission probabilities:
  \[
  \hat{P}(w_j | t_i) = \frac{\langle C(w_j, t_i) \rangle}{\langle C(t_i) \rangle}
  \]

- Our estimate for the initial state probabilities:
  \[
  \pi(t_i) = \frac{\langle C(\text{Tag of first word} = t_i) \rangle}{\text{Number of sentences}}
  \]

The Forward algorithm

\( \text{trellis}[n][i] \) stores the probability mass of all tag sequences \( t^{(1)}...t^{(n)} \) that end in tag \( t_i \) for the prefix \( w^{(1)}...w^{(n)} \)

\[
\text{trellis}[n][i] = \sum_{t^{(1)}...t^{(n-1)}} \left[ \frac{P(w^{(1)}...w^{(n)}) t^{(1)}...t^{(n-1)}, t^{(n)} = t_i} {P(w^{(n)})} \right]
\]

\[
= \sum_j \left[ \text{trellis}[n-1][j] \times P(t_i | t_j) \right] \times P(w^{(n)} | t_i)
\]

\[
= \sum_j \left[ \sum_{t^{(1)}...t^{(n-2)}} P(w^{(1)}...w^{(n-1)}, t^{(1)}...t^{(n-2)}, t^{(n-1)} = t_j) \times P(t_i | t_j) \right] \times P(w^{(n)} | t_i)
\]

| \( t_1 \) | \( \sum_{t^{(1)}...t^{(n-2)}} P(w^{(1)}...w^{(n-1)}, t^{(1)}...t^{(n-2)}, t^{(n-1)} = t_1) \) | \( P(t_1 | t_0) \) |
| --- | --- | --- |
| ... | ... | ... |
| \( t_i \) | \( \sum_{t^{(1)}...t^{(n-2)}} P(w^{(1)}...w^{(n-1)}, t^{(1)}...t^{(n-2)}, t^{(n-1)} = t_i) \) | \( P(t_i | t_{i-1}) \) |
| ... | ... | ... |
| \( t_{T} \) | \( \sum_{t^{(1)}...t^{(n-2)}} P(w^{(1)}...w^{(n-1)}, t^{(1)}...t^{(n-2)}, t^{(n-1)} = t_{T}) \) | \( P(t_{T} | t_{T-1}) \) |

Last step: computing \( P(w) \):

\[
P(w^{(1)}...w^{(N)}) = \sum_j \text{trellis}[N][j]
\]

Expected counts

Emission probabilities with observed counts \( C(w, t) \)

\[
P(w | t) = \frac{C(w, t)}{\langle C(t) \rangle} = \frac{C(w, t)}{\sum_{w'} C(w', t)}
\]

Emission probabilities with expected counts \( \langle C(w, t) \rangle \)

\[
P(w | t) = \frac{\langle C(w, t) \rangle}{\langle C(t) \rangle} = \frac{\langle C(w, t) \rangle}{\sum_{w'} \langle C(w', t) \rangle}
\]

\( \langle C(w, t) \rangle \): How often do we expect to see word \( w \) with tag \( t \) in our training data (under a given HMM)?

We know how often the word \( w \) appears in the data, but we don't know how often it appears with tag \( t \)

We need to sum up \( \langle C(w^{(i)} = w, t) \rangle \) for any occurrence of \( w \)

We can show that \( \langle C(w^{(i)} = w, t) \rangle = P(t^{(i)} = t | w) \)

(NB: Transition counts \( \langle C(t^{(i)} = t, t^{(i+1)} = t') \rangle \) work in a similar fashion)
**Forward-Backward:** \( P(t^{(i)} = t \mid w^{(1)}..(N)) \)

\[
P(t^{(i)} = t \mid w^{(1)}..(N)) = P(t^{(i)} = t, w^{(1)}..(N)) / P(w^{(1)}..(N))
\]

Due to HMM’s independence assumptions:
\[
P(t^{(i)} = t \mid w^{(1)}..(N)) = P(t^{(i)} = t, w^{(1)}..(i)) \times P(w^{(i+1)}..(N) \mid t^{(i)} = t)
\]

The forward algorithm gives \( P(w^{(1)}..(N)) = \sum_t P(w^{(1)}..(N)) \)

**Forward trellis:** \( \text{forward}[i][t] = P(t^{(i)} = t, w^{(1)}..(i)) \)

Gives the total probability mass of the prefix \( w^{(1)}..(i) \), summed over all tag sequences \( t^{(i)}..(i) \) that end in tag \( t^{(i)} = t \)

**Backward trellis:** \( \text{backward}[i][t] = P(w^{(i+1)}..(N) \mid t^{(i)} = t) \)

Gives the total probability mass of the suffix \( w^{(i+1)}..(N) \), summed over all tag sequences \( t^{(i+1)}..(N) \), if we assign tag \( t^{(i)} = t \) to \( w^{(i)} \)

**The Backward algorithm**

The backward trellis is filled from right to left.

\( \text{backward}[i][t] \) provides \( P(w^{(i+1)}..(N) \mid t^{(i)} = t) \)

NB: \( \sum_t \text{backward}[1][t] = P(w^{(i+1)}..(N)) = \sum_t \text{forward}[N][t] \)

**Initialization (last column):**

\( \text{backward}[N][t] = 1 \)

**Recursion (any other column):**

\[
\text{backward}[i][t] = \sum_{t'} P(t' \mid t) \times P(w^{(i+1)} \mid t') \times \text{backward}[i+1][t']
\]

**The importance of tag dictionaries**

Forward-Backward assumes that each tag can be assigned to any word.

No guarantee that the learned HMM bears any resemblance to the tags we want to get out of a POS tagger.

A **tag dictionary** lists the possible POS tags for words.

Even a partial dictionary that lists only the tags for the most common words and contains at least a few words for each tag provides enough constraints to get significantly closer to a model that produces linguistically correct (and hence useful) POS tags.