Lecture 9:
Sequence Labeling

Julia Hockenmaier
juliahm@illinois.edu
3324 Siebel Center
Friday’s key concepts (I)

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
Expected counts

Emission probabilities with *observed counts* $C(w, t)$

$$P(w \mid t) = \frac{C(w, t)}{\sum w: C(t)} = \frac{C(w, t)}{\sum w: \ C(w', t)}$$

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

$$P(w \mid t) = \frac{\langle C(w, t) \rangle}{\sum w' \langle C(t) \rangle} = \frac{\langle C(w, t) \rangle}{\sum w' \langle C(w', t) \rangle}$$

$\langle C(w, t) \rangle$: 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 $\langle C(w^{(i)}=w, t) \rangle = P(t^{(i)}=t \mid w)$

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

\[
P( t^{(i)}=t \mid \mathbf{w}(1)\ldots(N) ) = \frac{P( t^{(i)}=t, \mathbf{w}(1)\ldots(N) )}{P(\mathbf{w}(1)\ldots(N))}
\]

\( \mathbf{w}(1)\ldots(N) = \mathbf{w}(1)\ldots(i)\mathbf{w}(i+1)\ldots(N) \)

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

The forward algorithm gives \( P(\mathbf{w}(1)\ldots(N)) = \sum_t \text{forward}[N][t] \)

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

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

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

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

The backward trellis is filled from right to left.

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

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):

\[ \text{backward}[i][t] = \sum_{t'} P(t' | t) \times P(w^{(i+1)} | t') \times \text{backward}[i+1][t'] \]
How do we compute $\langle C(t_i) \mid w_j \rangle$?

$$
\begin{array}{ccccccc}
& w^{(1)} & \cdots & w^{(i-1)} & w^{(i)} & w^{(i+1)} & \cdots & w^{(N)} \\
\hline
t_1 & & & & & & & \\
\vdots & & & & & & & \\
t & & & & & & & \\
\vdots & & & & & & & \\
t_T & & & & & & & \\
\end{array}
$$

\[ \langle C(t, w^{(i)}) \mid w \rangle = P(t^{(i)} = t, w)/P(w) \]

with

\[ P(t^{(i)} = t, w) = \text{forward}[i][t] \text{ backward}[i][t] \]

\[ P(w) = \sum_t \text{forward}[N][t] \]
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.

<table>
<thead>
<tr>
<th></th>
<th>DT</th>
<th>back</th>
<th>JJ, NN, VB, VBP, RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>an</td>
<td>DT</td>
<td>bank</td>
<td>NN, VB, VBP</td>
</tr>
<tr>
<td>and</td>
<td>CC</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>America</td>
<td>NNP</td>
<td>zebra</td>
<td>NN</td>
</tr>
</tbody>
</table>
Sequence labeling
Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

**Task:** assign POS tags to words
Pierre Vinken, 61 years old, will join IBM’s board as a nonexecutive director Nov. 29.

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

Task: identify all non-recursive NP chunks
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

```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] ’s [NP board] 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_O 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 ._O
```
Shallow parsing

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

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

[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.

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].
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**, **CD** 61 **CD** years **NNS** old **JJ**, **MD** will **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**
Graphical models for sequence labeling
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:

- $P(X) = \text{\textbullet} \ x$

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

- $P(Y) \ P(X \mid Y) = \text{\textbullet} \ y \to \ x$

- $P(Y) \ P(Z) \ P(X \mid Y, Z) = \text{\textbullet} \ y \to \ x \text{\textbullet} \ z \to \ x$

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

- $P(Y) \ P(X \mid Y)$ with $Y$ hidden and $X$ observed = $\text{\textbullet} \ y \to \ x$
HMMs as graphical models

HMMs are \textbf{generative} models of the observed input string \( w \)
They ‘generate’ \( w \) with
\[
P(w, t) = \prod_i P(t(i) | t(i-1))P(w(i) | t(i))
\]
When we use an HMM to tag, we observe \( w \), and need to find \( t \)
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 \( P(t \mid w) \) 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,…)  
– These features may also help us deal with unknown words  
– But these features may not be independent  
  (e.g. they are overlapping)

Modeling $P(t | w)$ 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

$$P(t \mid w) = \prod_i P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$$

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

There are different ways to model $P( c \mid x)$. MEMMs and CRFs are based on logistic regression
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 $x$: $score(x,c) = \sum_i w_{ic} f_i(x)$

This score could be negative, so we exponentiate it: $score(x,c) = \exp(\sum_i w_{ic} f_i(x))$

To get a probability distribution over all classes $c$, we renormalize these scores:

$P(c \mid x) = score(x,c) / \sum_j score(x,c_j)$

$= \exp(\sum_i w_{ic} f_i(x)) / \sum_j \exp(\sum_i w_{ij} f_i(x))$
Learning: finding $\mathbf{w}$

Learning = finding weights $\mathbf{w}$

We use conditional maximum likelihood estimation (and standard convex optimization algorithms) to find/learn $\mathbf{w}$

(for more details, attend CS446 and CS546)

The conditional MLE training objective:

Find the $\mathbf{w}$ that assigns highest probability to all observed outputs $c_i$ given the inputs $x_i$

$$\hat{\mathbf{w}} = \arg \max_{\mathbf{w}} \prod_i P(c_i | x_i, \mathbf{w})$$
Terminology

Models that are of the form
\[
P(c | x) = \frac{\text{score}(x,c)}{\sum_j \text{score}(x,c_j)}
= \exp( \sum_i w_{ic} f_i(x)) / \sum_j \exp( \sum_i w_{ij} f_i(x))
\]

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 \( \sum_j \exp( \sum_i w_{ij} f_i(x)) \) is also called the partition function and is often abbreviated as \( Z \)
**Maximum Entropy Markov Models**

MEMMs use a MaxEnt classifier for each $P(t^{(i)} | w^{(i)}, t^{(i-1)})$:

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 $t_k$:

$$P(t^{(i)} = t_k | t^{(i-1)}, w^{(i)}) = \frac{\exp(\sum_j \lambda_{jk} f_j(t^{(i-1)}, w^{(i)}))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t^{(i-1)}, w^{(i)}))}$$
Viterbi for MEMMs

trellis[n][i] stores the probability of the most likely (Viterbi) tag sequence \( t^{(1)} \ldots t^{(n)} \) that ends in tag \( t_i \) for the prefix \( w^{(1)} \ldots w^{(n)} \). Remember that we do not generate \( w \) in MEMMs. So:

\[
trellis[n][i] = \max_{t^{(1)} \ldots (n-1)} \left[ P(t^{(1)} \ldots (n-1), t^{(n)} = t_i | w^{(1)} \ldots (n)) \right] \\
= \max_j \left[ \text{trellis}[n-1][j] \times P(t_i | t_j, w^{(n)}) \right] \\
= \max_j \left[ \max_{t^{(1)} \ldots (n-2)} \left[ P(t^{(1)} \ldots (n-2), t^{(n-1)} = t_j | w^{(1)} \ldots (n-1)) \right] \times P(t_i | t_j, w^{(n)}) \right]
\]
Today’s key concepts

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

Discriminative models:
- Maximum Entropy classifiers
- MEMMs