Lecture 9: Sequence Labeling

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>
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: assign POS tags to words

Noun phrase (NP) chunking

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] .

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
```

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_B-NP years_I-NP 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
```

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)

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 years _NNS old _JJ , _MD 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
HMMs as graphical models

HMMs are 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 \( t \)

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:
- \( P(X) = \)

Arrows represent dependencies (they define what other random variables the current node is conditioned on)
- \( P(Y) P(X | Y) = \)
- \( P(Y) P(Z) P(X | Y, Z) = \)

Shaded nodes represent observed variables.
White nodes represent hidden variables
- \( P(Y) P(X | Y) \) with \( Y \) hidden and \( X \) observed =

Models for sequence labeling

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

argmax \( P(t | w) \)

Generative models use Bayes Rule:
\[
\argmax_t P(t | w) = \argmax_t \frac{P(t, w)}{P(w)} = \argmax_t P(t, w) = \argmax_t P(t)P(w | t)
\]

Discriminative (conditional) models model \( P(t | w) \) directly
Advantages of discriminative models

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

Modeling $P(w \mid t)$ 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 $P(t \mid w)$ 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.

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 $c^* = \text{argmax}_i P(c_i \mid x)$
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 $x$:

$$\text{score}(x, c) = \sum_i w_{ic} f_i(x)$$

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:

$$P(c | x) = \frac{\text{score}(x, c)}{\sum_j \text{score}(x, c_j)} = \frac{\exp(\sum_i w_{ic} f_i(x))}{\sum_j \exp(\sum_i w_{ij} f_i(x))}$$

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 $x_i$

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

Terminology

Models that are of the form

$$P(c | x) = \frac{\text{score}(x, c)}{\sum_j \text{score}(x, c_j)} = \frac{\exp(\sum_i w_{ic} f_i(x))}{\sum_j \exp(\sum_i w_{ij} f_i(x))}$$

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 $P(t(i) \mid w(i), t(i-1))$:

$$P(t(i) = t_k \mid t(i-1), w(i)) = \frac{\exp(\sum_j \lambda_{jk} f_j(t(i-1), w(i)))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t(i-1), w(i)))}$$

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 $t_k$:

$$P(t(i) = t_k \mid t(i-1), w(i)) = \frac{\exp(\sum_j \lambda_{jk} f_j(t(i-1), w(i)))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t(i-1), w(i)))}$$

Viterbi for MEMMs

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

Remember that we do not generate $w$ in MEMMs. So:

$$\text{trellis}[n][i] = \max_{t^{(1)}...(n-1)} [ P(t^{(1)}...(n-1), t^{(n)}=t_i \mid w^{(1)}...(n)) ]$$

$$= \max_j [ \text{trellis}[n-1][j] \times P(t_i \mid t_j, w^{(n)}) ]$$

$$= \max_j [ \max_{t^{(1)}...(n-2)} [ P(t^{(1)}...(n-2), t^{(n-1)}=t_j \mid w^{(1)}...(n-1)) ] \times P(t_i \mid t_j, w^{(n)}) ]$$

Today’s key concepts

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

Discriminative models:
- Maximum Entropy classifiers
- MEMMs