Sequence Labelling and Part-of-Speech Tagging

COM6513 Natural Language Processing

Nikos Aletras
n.aletras@sheffield.ac.uk

@nikaletras

Computer Science Department

Week 4
Spring 2021

The University Of Sheffield.

In the previous lecture...

- Our first **sequence modelling** problem: **Language Modelling**

In this lecture...

- What about if we want to **assign a label to each word in a sequence**?

In this lecture...

- What about if we want to **assign a label to each word in a sequence**?
- Sequence labelling!

In this lecture...

- What about if we want to **assign a label to each word in a sequence**?
- Sequence labelling!
- Applications?

Applications

- Part-of-Speech (POS) Tagging

$$
\begin{aligned}
(\mathbf{x}, \mathbf{y}) = (&[I, studied, in, Sheffield], \\
&[Pronoun, Verb, Preposition, ProperNoun])
\end{aligned}
$$

Applications

- Part-of-Speech (POS) Tagging

$$
\begin{aligned}
(\mathbf{x}, \mathbf{y}) = ([&I, studied, in, Sheffield], \\
&[Pronoun, Verb, Preposition, ProperNoun])
\end{aligned}
$$

- Named Entity Recognition

$$
\begin{aligned}
(\mathbf{x}, \mathbf{y}) = ([&Giannis, Antetokounmpo, plays, for, the, Bucks], \\
&[Person, Person, NotEnt, NotEnt, NotEnt, Org])
\end{aligned}
$$

- Machine Translation (reconstruct word alignments)

$$
\begin{aligned}
(\mathbf{x}, \mathbf{y}) = ([&la, maison, bleu], \\
&[the, house, blue])
\end{aligned}
$$

We will use POS tagging as a running example

Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	’s	news
determiner	noun	verb	preposition	noun	possessive	noun

Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	’s	news
determiner	noun	verb	preposition	noun	possessive	noun

What could they be useful for?

Parts of Speech (POS)

Label words according to their syntactic function in a sentence:

The	results	appear	in	today	's	news
determiner	noun	verb	preposition	noun	possessive	noun

What could they be useful for?

- text classification
- language modelling
- syntactic parsing
- named entity recognition
- question answering

PoS Tags

- Open class:
  nouns, verbs, adjevtives
- Closed class:
  determiners, prepositions, conjunctions, etc

PoS definitions

- Most research uses the Penn Treebank PoS tag set
- Includes 45 tags making distinctions between:
  - verbs in active vs past tense
  - nouns in singular vs plural number
  - etc.

PoS definitions

- Most research uses the Penn Treebank PoS tag set
- Includes 45 tags making distinctions between:
  - verbs in active vs past tense
  - nouns in singular vs plural number
  - etc.
- Penn Tree Bank inspired by English. Recent work has focused on the Universal PoS tag set:
  - 17 coarse tags: one noun class, one verb class, etc.
  - developed considering 22 languages

Sequence labelling: Problem Setup

Data consists of word sequences with label sequences:

$$
\begin{aligned}
D_{train} &= \{(\mathbf{x}^1, \mathbf{y}^1)...(\mathbf{x}^M, \mathbf{y}^M)\} \\
\mathbf{x}^m &= [x_1, ...x_N] \\
\mathbf{y}^m &= [y_1, ...y_N]
\end{aligned}
$$

Learn a model f that predicts the best label sequence:

$$
\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} f(\mathbf{x}, \mathbf{y}) \tag{1}
$$

$\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}$ is the set of all possible combinations of label sequences and $\mathcal{Y} = \{A, B, C...\}$ are the possible classes for each word.

Could we use a dictionary-based model?

$$\{\textit{the} : \textit{determiner}, \textit{can} : \textit{modal}, \textit{fly} : \textit{verb}\}$$

Could we use a dictionary-based model?

$$\{the : determiner, can : modal, fly : verb\}$$

Yes, but the same word can have different tags in different contexts.

I	can	fly
pronoun	modal	verb

vs:

I	can	fly
pronoun	verb	noun

can and 11.5% of the words in the Brown corpus have more than one tag

Can we use a Markov model?

Use tags $\mathbf{y}$ instead of words:

$$P(\mathbf{y}) = \prod_{n=1}^{N} P(y_n | y_{n-1})$$

Can we use a Markov model?

Use tags **y** instead of words:

$$P(\mathbf{y}) = \prod_{n=1}^{N} P(y_n | y_{n-1})$$

Our training data should be able to tell us that tagging is unlikely:

I	can	fly
pronoun	modal	noun

Can we use a Markov model?

Use tags **y** instead of words:

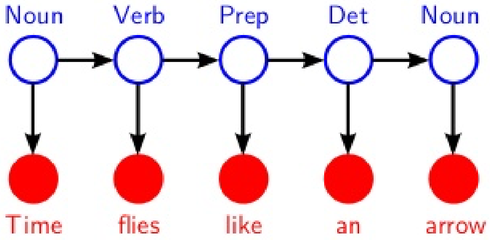
$$P(\mathbf{y}) = \prod_{n=1}^{N} P(y_n | y_{n-1})$$

Our training data should be able to tell us that tagging is unlikely:

I	can	fly
pronoun	modal	noun

What about the words? We will get whe same N-tag long sequence for any sentence!

Hidden Markov Model (HMM)



- Labels y_i (i.e. PoS tags) are hidden states emitting words.
- Assumptions:
  - 1st order Markov among the POS tags (current tag depends only on previous tag) verb noun
  - Each word only depends on its POS tag

HMM: Derivation

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{y}|\mathbf{x}) \quad \text{(Bayes rule)}$$

HMM: Derivation

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{y}|\mathbf{x}) \quad \text{(Bayes rule)}$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{\cancel{P(\mathbf{x})}} \quad \text{(word probabilities are constant)}$$

HMM: Derivation

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{y}|\mathbf{x}) \quad \text{(Bayes rule)}$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{\cancel{P(\mathbf{x})}} \quad \text{(word probabilities are constant)}$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \quad \text{(1st order Markov)}$$

HMM: Derivation

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{y}|\mathbf{x}) \quad \text{(Bayes rule)}$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{\cancel{P(\mathbf{x})}} \quad \text{(word probabilities are constant)}$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \quad \text{(1st order Markov)}$$

$$\hat{\mathbf{y}} \approx \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \prod_{n=1}^{N} P(x_n|y_n)P(y_n|y_{n-1})$$

HMM: Training

- Maximum likelihood estimation (i.e. counts!):

  $P(y_n|y_{n-1}) = \frac{c(y_n, y_{n-1})}{c(y_{n-1})}$ (transition probabilities)

  $P(x_n|y_n) = \frac{c(x_n, y_n)}{c(y_n)}$ (emission probabilities)

HMM: Training

- Maximum likelihood estimation (i.e. counts!):

  $P(y_n|y_{n-1}) = \frac{c(y_n,y_{n-1})}{c(y_{n-1})}$ (transition probabilities)

  $P(x_n|y_n) = \frac{c(x_n,y_n)}{c(y_n)}$ (emission probabilities)

- We can easily compute counts $c(\cdot)$ using a labelled corpus (pairs of words-POS tags).

HMM: Example

x = [START, I, can, fly, END]
y = [START, PPSS, MD, NN, END]

$$\hat{\mathbf{y}} \approx \underset{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}}{\arg\max} \prod_{n=1}^{N} P(x_n|y_n) P(y_n|y_{n-1})$$

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) = & P(I|PPSS)P(PPSS|START) \\ & P(can|MD)P(MD|PPSS) \\ & P(fly|NN)P(NN|MD) \end{aligned}$$

Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \prod_{n=1}^{N} P(x_n|y_n)P(y_n|y_{n-1})$$

- Just enumerate all possible tag sequences?

Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \prod_{n=1}^{N} P(x_n | y_n) P(y_n | y_{n-1})$$

- Just enumerate all possible tag sequences?
Intractable. We would need to evaluate $|\mathcal{Y}|^{\mathcal{N}}$ sequences!

Decoding/Inference

- So we have everything we need to decode/infer the most likely tag sequence for a sentence:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \prod_{n=1}^{N} P(x_n|y_n)P(y_n|y_{n-1})$$

- Just enumerate all possible tag sequences?
  Intractable. We would need to evaluate $|\mathcal{Y}|^{\mathcal{N}}$ sequences!
- We will see later how to decode efficiently!

HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small

HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small
- Smoothing:
  - what happens when we have unseen word/tags or tag-tag combinations?

HMMs: Some extra points

- Higher order HMMs:
  - longer contexts, more expensive inference
  - benefits are usually small
- Smoothing:
  - what happens when we have unseen word/tags or tag-tag combinations?
    Use methods we learned in the language modeling lecture!

$$\frac{c \quad +k}{c \quad +|V|k}$$

HMMs: Limitations

- They generate probabilities for words and labels, we just want labels
- No overlapping features (e.g. unigrams+bigrams)
- No subword features (e.g. suffixes)

Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!

Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step $\rightarrow$ Conditional Random Fields

Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step $\rightarrow$ Conditional Random Fields

Conditional Random Fields: Extend LR for Sequence Labelling

- Logistic regression can also provide probabilities but supports more flexible representations!
- Given a word, a candidate label of the current and the label of the previous word, predict the most likely label for that word using a (multi-class LR) in each time step $\rightarrow$ Conditional Random Fields
- CRF paper more than 11K citations since 2001, 10 year test of time award at ICML conference

Conditional Random Fields

Decompose the per sentence $\mathbf{x} = [x_1, ...x_N]$ prediction:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} f(\mathbf{x}, \mathbf{y})$$

into each word x_n:

$$\hat{y}_n = \arg\max_{y \in \mathcal{Y}} f(x_n; y_{n-1}, n) = \arg\max_{y \in \mathcal{Y}} \mathbf{w}^y \phi(x_n, y_{n-1}, n)$$

Conditional Random Fields

Decompose the per sentence $\mathbf{x} = [x_1, ...x_N]$ prediction:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} f(\mathbf{x}, \mathbf{y})$$

into each word x_n:

$$\hat{y}_n = \arg\max_{y \in \mathcal{Y}} f(x_n; y_{n-1}, n) = \arg\max_{y \in \mathcal{Y}} \mathbf{w}^y \phi(x_n, y_{n-1}, n)$$

- How to construct a feature vector $\phi(x_n, y_n, y_{n-1}, n)$?

CRF: Feature Vectors

- $\phi_1(x_n, y_n, y_{n-1}, n) = 1$ if $y_n = ADVERB$ and the n-th word ends in "-ly"; 0 otherwise.
  "usually","casually"
- $\phi_2(x_n, y_n, y_{n-1}, n) = 1$ if $n = 1,\ y_n = VERB$, and the sentence ends in a question mark; 0 otherwise.
  "Is it true?"
- etc.

CRF: Inference

The normalisation factor has to score all possible label sequences for all sentences, so it is ignored:

$$\arg\max_{\mathbf{y}\in\mathcal{Y}^{\mathcal{N}}} P_{CRF}(\mathbf{y}|\mathbf{x};\mathbf{w}) = \arg\max_{\mathbf{y}\in\mathcal{Y}^{\mathcal{N}}} \sum_{n=1}^{N} \mathbf{w}\cdot\phi(y_n, y_{n-1}, \mathbf{x}, n)$$

CRF: Training

- Training by minimising the negative log-likelihood objective:

$$\mathbf{w} = \arg\min_{\mathbf{w} \in \Re^d} \sum_{m=1}^{M} -\log P_{CRF}(\mathbf{y^m} | \mathbf{x^m}; \mathbf{w})$$

- using **Stochastic Gradient Descent**

Decoding with Viterbi

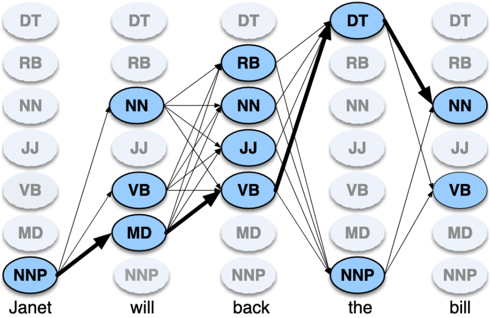
- Enumerating all possible tag sequences in HMM and CRF is intractable!

Decoding with Viterbi

- Enumerating all possible tag sequences in HMM and CRF is intractable!
- Dynamic programming: store and re-use calculations
- Possible due to independence assumptions
- Keep track of the highest probability to reach each PoS tag for each word and how we got there

Decoding with Viterbi

$|Y|$



N

Viterbi: Data structures

- **Viterbi score matrix** $V^{|\mathcal{Y}| \times N}$:

Viterbi: Data structures

- **Viterbi score matrix** $V^{|\mathcal{Y}| \times N}$:
  - Tag set $\mathcal{Y}$, sentence $\mathbf{x} = [x_1, ...x_N]$

Viterbi: Data structures

- **Viterbi score matrix** $V^{|\mathcal{Y}| \times N}$:
  - Tag set $\mathcal{Y}$, sentence $\mathbf{x} = [x_1, ... x_N]$
  - each cell contains the highest prob. for word n with tag y

Viterbi: Data structures

- **Viterbi score matrix** $V^{|\mathcal{Y}| \times N}$:
  - Tag set $\mathcal{Y}$, sentence $\mathbf{x} = [x_1, ...x_N]$
  - each cell contains the highest prob. for word n with tag y
  - 1st order Markov: only depends on the previous tag y_{n-1}
    $V[y, n] = \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n | x_n, y_{n-1})$

Viterbi: Data structures

- **Backpointer matrix** $backptr^{|\mathcal{Y}| \times N}$:

Viterbi: Data structures

- **Backpointer matrix** $backptr^{|\mathcal{Y}| \times N}$:
  - instead of the max score, keep the previous tag that got it

Viterbi: Data structures

- **Backpointer matrix** $backptr^{|\mathcal{Y}| \times N}$:
  - instead of the max score, keep the previous tag that got it
  - *argmax* instead of *max*
    $backptr[y, n] = \arg\max_{y' \in \mathcal{Y}} V[y', n-1] \times P(y|y') \times P(x_n|y)$

Viterbi algorithm

Input: *word sequence* $\mathbf{x} = [x_1, ..., x_N]$,

$P(y_n|x_n, y_{n-1})$ *probs*

set matrix $V^{|\mathcal{Y}| \times N} = 1$

for $n = 1$ **to** N **do**

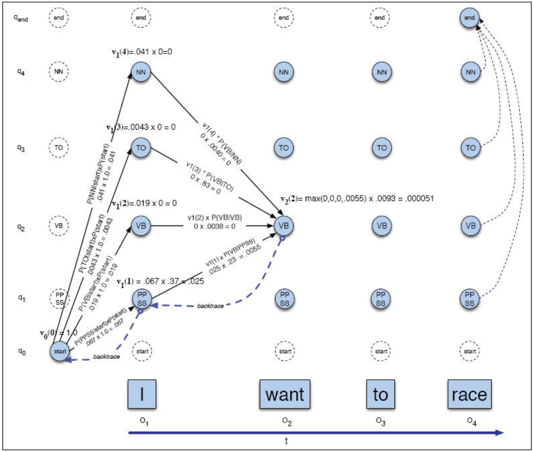
  for $y \in \mathcal{Y}$ **do**

$$V[y, n] = \max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n|x_n, y_{n-1})$$

$$backptr[y, n] = \arg\max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, n-1] \times P(y_n|x_n, y_{n-1})$$

$$backptr[None, N+1] = \arg\max_{y_{n-1} \in \mathcal{Y}} V[y_{n-1}, N] \times P(None|y_{n-1})$$

Viterbi diagram



Break the large arg max into smaller ones, left-to-right (**dynamic programming**)

Beam Search: Inexact Decoding

- Viterbi performs exact search (under assumptions) by evaluating all options.

Beam Search: Inexact Decoding

- Viterbi performs exact search (under assumptions) by evaluating all options.
- Get faster by being inexact, i.e. avoid labelling some candidate sequences with **Beam Search**

Beam Search

- Do Viterbi, but keep only best k hypotheses at each step

Beam Search

- Do Viterbi, but keep only best k hypotheses at each step
- If beam size is 1, then we have greedy search

Beam Search

- Do Viterbi, but keep only best k hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster

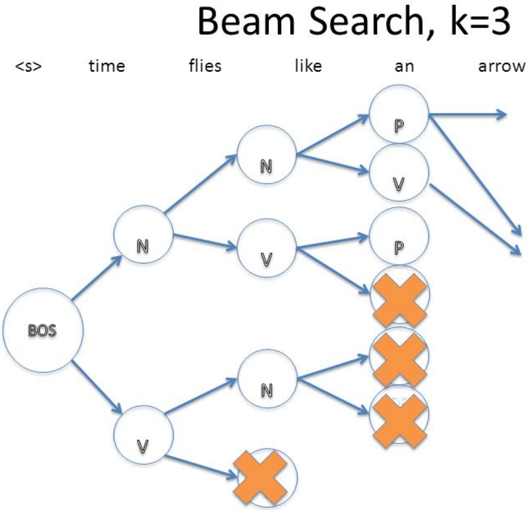
Beam Search

- Do Viterbi, but keep only best k hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster
- Beams must be of the same length to be comparable

Beam Search

- Do Viterbi, but keep only best k hypotheses at each step
- If beam size is 1, then we have greedy search
- Often beams less than 10 get close to exact search, but much faster
- Beams must be of the same length to be comparable
- Attractive when we need complex feature functions i.e. avoid Markov assumptions)

Beam Search: Example



Beam Search: Algorithm

Input: *word sequence* $\mathbf{x} = [x_1, ..., x_N]$, *weights* $\mathbf{w}$

set beam $B = \{(\mathbf{y_{temp}} = [START], score = 0)\}$, *size* k

for $n = 1$ **to** N **do**

  $B' = \{\}$

  for $b \in B$ **do**

    for $y \in \mathcal{Y}$ **do**

      $B' = B' \cup ([b.\mathbf{y_{temp}}; y], P([b.\mathbf{y_{temp}}; y] | x_n))$

  $B = \text{TOP-k}(B')$

return $\text{TOP-1}(B)$

Bibliography

- Chapter 8 from Jurafsky and Martin
- Sections 7.1-7.4, 7.5.3 and Chapter 8 from Eisenstein
- This blog post on CRFs by Edwin Chen
- Tutorial on CRFs by Sutton and McCallum

Coming up next...

The best-studied, more complex than sequence labeling problem in NLP: **dependency parsing**