

## Part-of-speech tagging. Part 2.<sup>1</sup>

Victor Kitov

[v.v.kitov@yandex.ru](mailto:v.v.kitov@yandex.ru)

---

<sup>1</sup>With materials used from "Speech and Language Processing", D. Jurafsky and J. H. Martin.

## Usage of part of speech

- Estimation of HMM
  - both outputs and states are known in the training set.
  - $\prod_{k=1}^K p(X_k, Y_k | A, B, \pi) \rightarrow \max_{A, B, \pi}$  where
    - $K$  is the number of sentences,
    - $X_k$  is  $k$ -th sentence
    - $Y_k$  is corresponding tags sequence.
  - Emission and transition probabilities will be estimated with empirical frequencies.
- Application of HMM:
  - For given sentence  $X$ , recover sequence of tags  $Y$  using

$$\begin{aligned} \hat{Y} &= \arg \max_Y p(Y|X) = \arg \max_Y \frac{p(Y)p(X|Y)}{p(X)} \\ &= \arg \max_Y p(Y)p(X|Y) \end{aligned}$$

## Details of HMM application

- Generated word depends only on part-of-speech:

$$p(X|Y) = \prod_{n=1}^N p(x_n|y_n)$$

- Next tag depends only on previous tag:

$$p(Y) = p(y_1) \prod_{n=2}^N p(y_n|y_{n-1})$$

- Final estimation

$$\hat{Y} = \arg \max_Y \overbrace{p(y_1) \prod_{n=2}^N p(y_n|y_{n-1})}^{\text{transition}} \overbrace{\prod_{n=1}^N p(x_n|y_n)}^{\text{emission}}$$

- argmax is found with Viterbi algorithm.

Advanced use of HMM<sup>23</sup>

- Emission probability conditioned on 2 previous states:

$$p(Y) = p(y_1, y_2) \prod_{n=3}^N p(y_n | y_{n-1}, y_{n-2})$$

- state - position in 2 states, instead of 1.
- Transition probability is replaced with

$$p(Y) = \prod_{n=1}^{N+1} p(y_n | y_{n-1}, y_{n-2})$$

where  $y_0, y_{-1}$  are special «before sentence tags» and  $y_{N+1}$  is «after sentence tag».

- To estimate  $p(y_t | y_{t-1}, y_{t-2})$  with insufficient data use smoothing:

$$p(y_t | y_{t-1}, y_{t-2}) = \lambda_3 \hat{p}(y_t | y_{t-1}, y_{t-2}) + \lambda_2 \hat{p}(y_t | y_{t-1}) + \lambda_1 \hat{p}(y_t)$$

- Parameters  $\lambda_1, \lambda_2, \lambda_3$  can be set with cross validation or using heuristic «deleted interpolation» method.

---

<sup>2</sup>Scott T., Harper M. 1999. A Second-Order Hidden Markov Model for

## Suffix

- for unknown words we can deduce POS using suffix
- suffix - informative for POS tagging:
  - ...able: likely adjective, ...ed: likely past tense of verb, etc.
- so we estimate  $p(y|word[-k :])$ 
  - try to estimate this with maximal  $k = 10$
  - if for big  $k$  we have no statistics, we fallback to probability for smaller  $k$  (*backoff* method)
- in HMM we need to generate observable suffixes, so we use:

$$p(word[-k :]|y) = \frac{p(word[-k :])p(y|word[-k :])}{p(y)}$$

- these probabilities are estimated separately for capitalized and uncapitalized words.
- replace  $(y_i)$  with pair  $(y_i, \mathbb{I}[\text{word } i \text{ is capitalized}])$  to treat capitalized words differently.
  - doubles number of states

## MEMM

- Consider sentence  $x_1 \dots x_N$  with POS tags  $y_1 \dots y_N$ .
- HMM prediction

$$\hat{Y} = \arg \max_Y p(y_1) \prod_{n=2}^N p(y_n | y_{n-1}) \prod_{n=1}^N p(x_n | y_n)$$

- MEMM (maximum entropy Markov model<sup>4</sup>) prediction

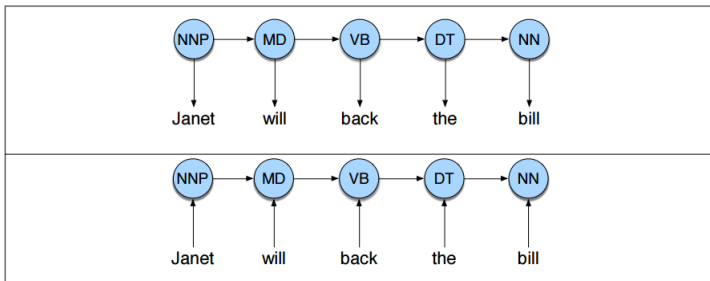
$$\hat{Y} = \arg \max_Y p(Y|X) = \arg \max_Y \prod_{n=1}^N p(y_n | x_n, y_{n-1})$$

---

<sup>4</sup>Maximum entropy name comes from the fact that most commonly logistic regression is used as classifier, which possesses the «maximum entropy» prediction properties.

## MEMM vs HMM

- Graphical structure of HMM (top) and MEMM (bottom):



- HMM - generative model and MEMM - discriminative.
  - it is easier to add new features to MEMM
  - in HMM need to add new features to  $p(x_n|y_n)$ 
    - harder to model

## Typical features in MEMM for predicting $y_n$

- neighbourhood words  $\langle w_{n+k} \rangle$ ,  $k = -2, -1, \dots, 2$ .
- neighbourhood word pairs  $\langle w_{n+k-1}, w_{n+k} \rangle$ ,  $k = 0, 1$ .
- previous tags:  $\langle y_{n-1} \rangle$ ,  $\langle y_{n-1} \rangle$ ,  $\langle y_{n-2} \rangle$ ,  $\langle y_{n-1}, y_{n-2} \rangle$
- tag&word combination  $\langle x_n, y_{n-1} \rangle$
- current word  $x_n$ :

contains a particular prefix (from all prefixes of length  $\leq 4$ )

contains a particular suffix (from all suffixes of length  $\leq 4$ )

contains a number

contains an upper-case letter

contains a hyphen

is all upper case

s word shape

s short word shape

is upper case and has a digit and a dash (like *CFC-12*)

is upper case and followed within 3 words by Co., Inc., etc.



## word-shape encoding of $x_n$ as feature

- rules:
  - letter->x
    - uppercase letter->X
    - digit->d
    - punctuation->punctuation (no change)
  - example:
    - U.S.A->X.X.X
    - FD-rsa18->XX-xxxdd
    - well-dressed->xxxx-xxxxxxx
- reduced word-shape: takes *word-shape encoding* symbols but without repetitions
  - examples:
    - FD-rsa18->X-xd
    - well-dressed->x-x
- rarely occurring (<5 times) shapes are not included to feature set.

## Application of MEMM

- For simplicity consider conditioning  $y_n$  only on  $X$  and  $y_{n-1}$ .
- Greedy MEMM decoding:

for  $n = 1, 2, \dots, N$ :  
 $y_n = \arg \max_y p(y|y_{n-1}, X)$

- fast
- makes greedy, local decisions
- cannot correct earlier decisions from later inconsistencies
- Viterbi algorithm gives a consistent sequence of predictions for whole sentence!

## Viterbi algorithm: forward pass

Assume  $p(y_t | \text{history}) = p(y_t | x_t, y_{t-1})$ . Definitions:

$$\varepsilon_t(i, X) := \max_{y_1, \dots, y_{t-1}} p(y_1 \dots y_{t-1} y_t = i | x_1 \dots x_t)$$

$$v_t(i, X) := \arg \max_j p(y_1 \dots y_{t-2}, y_{t-1} = j, y_t = i | x_1 \dots x_t)$$

Init:

$$\varepsilon_1(i, X) = p(y_1 = i | x_1) = \text{output of classifier}$$

For  $t = 1, \dots, T - 1$ :

$$\begin{aligned} \varepsilon_{t+1}(i, X) &= \max_{y_1 \dots y_{t-1} j} p(y_1 \dots y_{t-1} y_t = j, y_{t+1} = i | x_1 \dots x_t x_{t+1}) \\ &= \max_j \max_{y_1 \dots y_{t-1}} p(y_1 \dots y_{t-1} y_t = j | x_1 \dots x_{t+1}) p(y_{t+1} = i | y_1 \dots y_{t-1} y_t = j, x_1 \dots x_{t+1}) \\ &= \max_j \max_{y_1 \dots y_{t-1}} p(y_1 \dots y_{t-1} y_t = j | x_1 \dots x_t) p(y_{t+1} = i | y_t = j, x_{t+1}) \\ &= \max_j \varepsilon_t(j, X) p(y_{t+1} = i | y_t = j, x_{t+1}) \end{aligned}$$

$$v_{t+1}(i, X) = \arg \max_j \varepsilon_t(j, X) p(y_{t+1} = i | y_t = j, x_{t+1})$$

## Viterbi algorithm: backward pass

## Definitions

$$y_1^*, \dots, y_T^* := \arg \max_{y_1, \dots, y_T} p(y_1, \dots, y_T | x_1, \dots, x_T)$$

$$\varepsilon_t(i, X) := \max_{y_1, \dots, y_{t-1}} p(y_1 \dots y_{t-1} y_t = i | x_1 \dots x_t)$$

$$v_t(i, X) := \arg \max_j p(y_1 \dots y_{t-2}, y_{t-1} = j | y_t = i, x_1 \dots x_t)$$

Init:

$$p^*(X) = \max_j \varepsilon_T(j, X)$$

$$y_T^*(X) = \arg \max_j \varepsilon_T(j, X)$$

For  $t = T - 1, T - 2, \dots, 1$  :

$$y_t^*(X) = v_{t+1}(y_{t+1}^*(X))$$

## Comments

- We could define  $\varepsilon_t(i, X) := \max_{y_1, \dots, y_{t-1}} p(y_1 \dots y_{t-1} y_t = i | x \dots x_{t+k})$  for some lookahead horizon  $k > 0$ .
- we could condition  $y_t$  on several states before  $y_{t-1}, y_{t-2}, \dots$
- We use left-to-right classification. Similarly we could use right-to-left classification and combine their outputs with meta-model.
- Also we could make several passes:
  - first pass: obtain most likely  $y_1, \dots, y_N$
  - second pass: make classification both on past *and future*.

Brill tagger<sup>5</sup>

- Generates a data-driven set of rules.
- Top rules for known words (in the dictionary):

|   | Change Tag |     |  |
|---|------------|-----|--|
| # | From       | To  | Condition                                    |
| 1 | NN         | VB  | Previous tag is <i>TO</i>                    |
| 2 | VBP        | VB  | One of the previous three tags is <i>MD</i>  |
| 3 | NN         | VB  | One of the previous two tags is <i>MD</i>    |
| 4 | VB         | NN  | One of the previous two tags is <i>DT</i>    |
| 5 | VBD        | VBN | One of the previous three tags is <i>VBZ</i> |

- Top rules for unknown words:

|    | Change Tag |     |   |
|----|------------|-----|---|
| #  | From       | To  | Condition                                     |
| 1  | NN         | NNS | Has suffix <b>-s</b>                          |
| 2  | NN         | CD  | Has character <b>.</b>                        |
| 3  | NN         | JJ  | Has character <b>-</b>                        |
| 4  | NN         | VBN | Has suffix <b>-ed</b>                         |
| 5  | NN         | VBG | Has suffix <b>-ing</b>                        |
| 6  | ??         | RB  | Has suffix <b>-ly</b>                         |
| 7  | ??         | JJ  | Adding suffix <b>-ly</b> results in a word.   |
| 8  | NN         | CD  | The word <b>\$</b> can appear to the left.    |
| 9  | NN         | JJ  | Has suffix <b>-al</b>                         |
| 10 | NN         | VB  | The word <b>would</b> can appear to the left. |

# Algorithm

- Brill tagger algorithm:

**INIT:** set most probable tag to each word

**REPEAT** while quality changes significantly:

  for each rule pattern  $R(\cdot)$

    for each rule pattern instantiation  $\gamma \in \Gamma$

      evaluate rule  $R(\gamma)$

  select most successful rule  $R^*(\gamma^*)$

  apply most successful rule  $R^*(\gamma^*)$  to training dataset

  add  $R^*(\gamma^*)$  to the end of selected rules list

**OUTPUT:** selected rules list

## Rule patterns

### Example of rule patterns for known and unknown words:

*Change tag a to tag b when:*

The preceding (following) word is tagged z.

The word two before (after) is tagged z.

One of the two preceding (following) words is tagged z.

One of the three preceding (following) words is tagged z.

The preceding word is tagged z and the following word is tagged w.

The preceding (following) word is tagged z and the word two before (after) is tagged w.

where  $a$ ,  $b$ ,  $z$  and  $w$  are variables over the set of parts of speech.

*Change the tag of an unknown word (from X) to Y if:*

Deleting the prefix (suffix)  $x$ ,  $|x| \leq 4$ , results in a word ( $x$  is any string of length 1 to 4).

The first (last) (1,2,3,4) characters of the word are  $x$ .

Adding the character string  $x$  as a prefix (suffix) results in a word ( $|x| \leq 4$ ).

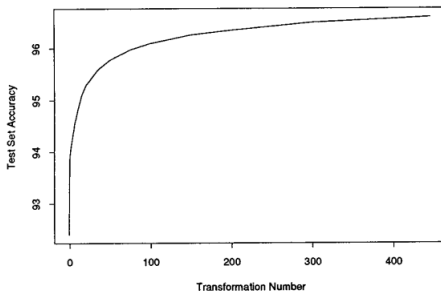
Word  $w$  ever appears immediately to the left (right) of the word.

Character  $z$  appears in the word.



## Comments

- Brill tagger gives comparative performance with HMM, but less than MEMM.
- Gives interpretable list of rules
- Accuracy on Wall Street Journal corpus 96.6%
- First rules give the most impact:



## Comments

- Brill tagger works very slow - needs to look through all rule patterns and all instantiations
- Possible improvements:
  - look only through those rule instantiations that improve at least 1 word tagging
  - use inverted index on rule conditions

## General sequence labelling

Sequence labelling: assign  $x_1 \dots x_N$  labels  $y_1, \dots, y_N$  where neighbouring labels are dependent.

Applications of sequence labelling:

- Part-of-speech tagging
- Speech recognition
- Handwriting recognition
- Video analysis (e.g. activity tagging)