# EMNLP
# Sequence Tagging II – Linear Models

**Yansong Feng**
**fengyansong@pku.edu.cn**

Institute of Computer Science and Technology
Peking University

February 19, 2017

# Recap: The HMM POS Tagger

- We represent
  - a sentence of any length $n$: $x_1, x_2, x_3, ...x_n$
  - its corresponding POS tag sequence; $y_1, y_2, y_3, ...y_n$
- We care the joint probability of a sentence and its POS tag sequence:

$$p(x_1, x_2, x_3, ...x_n, y_1, y_2, y_3, ...y_n)$$

(Generative Model)

- Then the most likely POS tag sequence for $x_1, x_2, x_3, ...x_n$:

$$\arg\max_{y_1 \dots y_n} p(y_1, y_2, y_3, ...y_n) p(x_1, x_2, x_3, ...x_n | y_1, y_2, y_3, ...y_n)$$

- Make Markov Assumptions (e.g., Trigram)

$$\arg\max_{y_1 \dots y_n} \prod_i p(y_i | y_{i-2}, y_{i-1}) \prod_i p(x_i | y_i)$$

# Elements in Our HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags
  - Co-occurrences of words and POS tags

## Elements in Our HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags $\rightarrow$ transition probabilities ( $p(y_n|y_{n-2}, y_{n-1})$)
  - Co-occurrences of words and POS tags $\rightarrow$ emission probabilities $(p(x_n|y_n))$

## Elements in Our HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags $\rightarrow$ transition probabilities ( $p(y_n|y_{n-2}, y_{n-1})$)
  - Co-occurrences of words and POS tags $\rightarrow$ emission probabilities ($p(x_n|y_n)$)

Anything else useful?

- if the current word ending with *ing*, *ed*, *se*, *ly*, *ical*, or ....
- if the previous word is *the*
- if the next word is *.*
- ...

# A Naive Way to Incorporate

…… many $p_{ML}$s

- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ing})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ed})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{se})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ly})$
- $p_{ML}(POS_{w_i} = \text{VB}w_i \text{ ending with } \textit{ical})$
- $p_{ML}(POS_{w_i} = \text{VB}w_{i-1} = \textit{the})$
- $p_{ML}(POS_{w_i} = \text{VB}w_{i+1} = \text{. (a period)})$
- …

# A Naive Way to Incorporate

...... many $p_{ML}$s

- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ing})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ed})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{se})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ly})$
- $p_{ML}(POS_{w_i} = \text{VB}w_i \text{ ending with } \textit{ical})$
- $p_{ML}(POS_{w_i} = \text{VB}w_{i-1} = \textit{ the})$
- $p_{ML}(POS_{w_i} = \text{VB}w_{i+1} = \text{ . (a period)})$
- ...

This gives you lots of $\lambda$s to tune.

## Another View: Features

**Features**: pieces of evidences describing some aspects of observed data $x$, usually with respect to the predicted label $y$

- computer vision
  - the shape, color, texture, size.....of an object
  - other objects nearby, relative postions
  - number of objects available
  - ...
- natural language process, e.g., POS tagging
  - the target word itself, prefix, suffix, capital or not, ...
  - context: words before/after the target, their morphology
  - number of those indications

# Features in NLP

**Features in NLP**: pieces of evidences describing some aspects of observed data $x$ with respect to the predicted label $y$, usually with the purpose of providing a conditional probability $p(y|x)$

# Features in NLP

**Features in NLP**: pieces of evidences describing some aspects of observed data $x$ with respect to the predicted label $y$, usually with the purpose of providing a conditional probability $p(y|x) \rightarrow$ discriminative models

# Features in NLP

**Features in NLP**: pieces of evidences describing some aspects of observed data $x$ with respect to the predicted label $y$, usually with the purpose of providing a conditional probability $p(y|x) \rightarrow$ discriminative models

## Often

- A feature is a function $f_i(x, y) \in \mathcal{R}$
- more often , it is a binary or indicator function
- for example,

$$f_i(x, y) = \begin{cases} 1 & \text{if } x = \text{Beijing and } y\text{=NNP} \\ 0 & \text{otherwise} \end{cases}$$

## Features in NLP

**Features in NLP**: pieces of evidences describing some aspects of observed data $x$ with respect to the predicted label $y$, usually with the purpose of providing a conditional probability $p(y|x) \rightarrow$ discriminative models

### Often

- A feature is a function $f_i(x, y) \in \mathcal{R}$
- more often , it is a binary or indicator function
- for example,

$$f_i(x, y) = \begin{cases} 1 & \text{if } x = \text{Beijing and } y\text{=NNP} \\ 0 & \text{otherwise} \end{cases}$$

- if we have $m$ aspects to describe an instance, i.e., $m$ features:
  - **a feature vector** for each instance, $(x, y)$
  - $[f_1(x, y), f_2(x, y), f_3(x, y), ..., f_m(x, y)]$
  - $[1, 0, 0, ...., 1, 0]$

## Features based Linear Models

Linear classifiers with the form like, $\lambda_i f_i(x, y)$

- need a linear function to map $f_i(x, y)$ to lable $y$
- possibly need a weight $\lambda_i$ for each feature $f_i(x, y)$
- then, for each possible label $y$ of instance $x$, we can compute a score:

$$score(x, y) = \sum_i \lambda_i f_i(x, y)$$

- the classifier should choose $y^*$:

$$y^* = \arg\max_y \sum_i \lambda_i f_i(x, y)$$

# Features based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- aspects: the target word, previous words, suffix, prefix, capitalized, ...
- curwd_Beijing_NNP, pre1word_love_NNP, pref_Be_NNP, cap_1_NNP, curwd_Beijing_VB, pref_Be_VB...
- for each possible labels (NNP, VB, DT, ...), coupled aspects with lables
- obtain $\lambda$s using some algorithm, $\lambda_{curwd\_Beijing\_NNP} = 10$, $\lambda_{pref\_Be\_NNP} = 5$, $\lambda_{cap\_1\_DT} = -10$, ...
- compute score(Beijing, NNP), score(Beijing, VB), score(Beijing, DT), ...
- choose the largest one: score(Beijing, NNP)
- tag *Beijing* with NNP

# Features based Linear Models: Algorithms

The key is to choose proper weights $\lambda$s for features

- the Perceptron algorithm
- Margin-based models (the Support Vector Machines, SVM)
- Exponential Models:
  - log-linear models, maximum entropy models, logistic models, ...
  - basically, produce a probabilistic model according to score$(x, y)$

$$p(y|x) = \frac{\exp score(x,y)}{\sum_{y'} \exp score(x,y')} = \frac{\exp \sum_i \lambda_i f_i(x,y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(x,y')}$$

  - numerator: positive score for label $y$
  - denominator: normalization

# Features based Linear Models: Algorithms

The key is to choose proper weights $\lambda$s for features

- the Perceptron algorithm(covered in this lecture)
- Margin-based models (the Support Vector Machines, SVM)
- Exponential Models:
  - log-linear models, maximum entropy models, logistic models, ...
  - basically, produce a probabilistic model according to score$(x, y)$

$$p(y|x) = \frac{\exp score(x,y)}{\sum_{y'} \exp score(x,y')} = \frac{\exp \sum_i \lambda_i f_i(x,y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(x,y')}$$

  - numerator: positive score for label $y$
  - denominator: normalization

# Features based Linear Models: Algorithms

The key is to choose proper weights $\lambda$s for features

- the Perceptron algorithm(covered in this lecture)
- Margin-based models (the Support Vector Machines, SVM)
- Exponential Models:
  - log-linear models, maximum entropy models, logistic models, ...
  - basically, produce a probabilistic model according to score$(x, y)$

$$p(y|x) = \frac{\exp score(x,y)}{\sum_{y'} \exp score(x,y')} = \frac{\exp \sum_i \lambda_i f_i(x,y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(x,y')}$$

  - numerator: positive score for label $y$
  - denominator: normalization
  - a powerful tool! (covered in later lectures)

# The Perceptron Algorithm

- Classic: Rosenblatt 1958
- Modern: Freund and Schapire 1999
  - proof for convergence
  - very competitive performances in classifications
- NLP: Michael Collins 2002, 2004, .....
  - modificatioins with respect to NLP applications
  - serves as alternative parameter estimation methods for many ML models
  - You SHOULD read at least the 2002 paper

# A Variant of The Perceptron Algorithm

- **Inputs**:
  - Training set $(x_k, y_k)$ for $k = 1, 2, ..., n$
  - $x_k$ the data, and $y_k$ the label
- Initialization:
  - $\lambda = [0, 0, 0....]$, $T$
- Define:
  - follow Collins: GEN enumerates possible candidate lable $y$s for data $x$
  - $z = \arg\max_{y \in GEN(x)} \sum_i \lambda_i f_i(x, y)$
- Loop:
  - For $t = 1, 2, 3..., T$, $k = 1, 2, 3, ..., n$
    compute $z_k = \arg\max_{y \in GEN(x_k)} \sum_i \lambda_i f_i(x_k, y)$
    update $\lambda$s
    - if $z_k \neq y_k$: $\lambda = \lambda + f(x_k, y_k) - f(x_k, z_k)$
- Output:
  - $\lambda$s

# A Variant of The Perceptron Algorithm

- **Inputs**:
  - Training set $(x_k, y_k)$ for $k = 1, 2, ..., n$
  - $x_k$ the data, and $y_k$ the label
- Initialization:
  - $\lambda = [0, 0, 0.....]$, $T$
- Define:
  - follow Collins: GEN enumerates possible candidate lable $y$s for data $x$
  - $z = \arg\max_{y \in GEN(x)} \sum_i \lambda_i f_i(x, y)$
- Loop:
  - For $t = 1, 2, 3..., T$, $k = 1, 2, 3, ..., n$
    compute $z_k = \arg\max_{y \in GEN(x_k)} \sum_i \lambda_i f_i(x_k, y)$ (Key: decode $z$)
    update $\lambda$s
    - if $z_k \neq y_k$: $\lambda = \lambda + f(x_k, y_k) - f(x_k, z_k)$
- Output:
  - $\lambda$s

## Structured Perceptron for POS Tagging (A simple case)

**training data:** *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- in a step during training:
  *China*/N *Mobile*/N ... *communication*/N *giant*/**??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**

## Structured Perceptron for POS Tagging (A simple case)

**training data:** *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- in a step during training:
  *China*/N *Mobile*/N ... *communication*/N *giant*/**??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, N, we extract $m$ features :
    - $f_1(x, y) = 1$ if current word is *giant* and $y = N$. $\rightarrow f_1(x, y) = 1$
    - $f_{11}(x, y) = 1$ if current word is *giant* and $y = ADJ$. $\rightarrow f_{11}(x, y) = 0$
    - $f_2(x, y) = 1$ if previous word is *the* and $y = N$. $\rightarrow f_2(x, y) = 0$
    - $f_{22}(x, y) = 1$ if previous word is *the* and $y = ADJ$. $\rightarrow f_{22}(x, y) = 0$
    - $f_3(x, y) = 1$ if sufix of current word is *ant* and $y = N$. $\rightarrow f_3(x, y) = 1$
    - $f_{33}(x, y) = 1$ if sufix of current word is *ant* and $y = ADJ$. $\rightarrow f_{33}(x, y) = 0$
    - ...
  - compute score$(giant, N) = \sum_i \lambda_i f_i(giant, N)$, score$(giant, ADJ)$, ...
  - choose the largest score$(giant, y)$ , e.g., ADJ

# Structured Perceptron for POS Tagging (A simple case)

**training data:** *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- in a step during training:
  *China*/N *Mobile*/N ... *communication*/N *giant*/**??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, N, we extract $m$ features :
    - $f_1(x, y) = 1$ if current word is *giant* and $y = N$. $\rightarrow f_1(x, y) = 1$
    - $f_{11}(x, y) = 1$ if current word is *giant* and $y = ADJ$. $\rightarrow f_{11}(x, y) = 0$
    - $f_2(x, y) = 1$ if previous word is *the* and $y = N$. $\rightarrow f_2(x, y) = 0$
    - $f_{22}(x, y) = 1$ if previous word is *the* and $y = ADJ$. $\rightarrow f_{22}(x, y) = 0$
    - $f_3(x, y) = 1$ if sufix of current word is *ant* and $y = N$. $\rightarrow f_3(x, y) = 1$
    - $f_{33}(x, y) = 1$ if sufix of current word is *ant* and $y = ADJ$. $\rightarrow f_{33}(x, y) = 0$
    - ...
  - compute score$(giant, N) = \sum_i \lambda_i f_i(giant, N)$, score$(giant, ADJ)$, ...
  - choose the largest score$(giant, y)$ , e.g., ADJ
- we can tag the whole sentence

# Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

# Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N
- we compare them, and find the differences:

# Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N
- we compare them, and find the differences:
- if necessary, we **update** the features related to the correct/wrong predictions
  - $\lambda^*_{f_1(x,y)} = \lambda_{f_1(x,y)} + 1$
  - $\lambda^*_{f_3(x,y)} = \lambda_{f_3(x,y)} + 1$
  - $\lambda^*_{f_{11}(x,y)} = \lambda_{f_{11}(x,y)} - 1$
  - $\lambda^*_{f_{33}(x,y)} = \lambda_{f_{33}(x,y)} - 1$
  - ...

# A Bit Complex

If we want to include features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_1 00(giant, N) = 1$
- $f_{101}(x, y) = 1$ if the previous two tags are *DT_N* and $y = N$. $\rightarrow$ $f_{101}(giant, N) = 1$
- ...
- we can not directly compute score$(giant, N)$, score$(giant, ADJ)$, ...

# A Bit Complex

If we want to include features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_1 00(giant, N) = 1$
- $f_{101}(x, y) = 1$ if the previous two tags are *DT_N* and $y = N$. $\rightarrow f_{101}(giant, N) = 1$
- ...
- we can not directly compute score$(giant, N)$, score$(giant, ADJ)$, ...
- we need to decode the best tag sequence for the whole sentence using Dynamic Programming

# A Bit Complex

If we want to include features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_1 00(giant, N) = 1$
- $f_{101}(x, y) = 1$ if the previous two tags are *DT_N* and $y = N$. $\rightarrow$ $f_{101}(giant, N) = 1$
- ...
- we can not directly compute score$(giant, N)$, score$(giant, ADJ)$, ...
- we need to decode the best tag sequence for the whole sentence using Dynamic Programming
  $\rightarrow$ **the Viterbi Algorithm**
-

$$\arg \max_{y \in GEN(x)} \sum_{w \in x} \sum_{i} \lambda_i f_i(\text{history}(w), y)$$

# Decoding: the Viterbi Algorithm

- for sentence of length $n$
- define the score of tag sequence $t_1, t_2, ...t_j$:
  score$(t_1, t_2, ...t_j) = \sum_{w \in x} \sum_i \lambda_i f'_i(w, t_{w-2}, t_{w-1}, t_w)$
- define the dynamic programming table
  $\pi(j, u, v) =$ maximum probability of a tag sequence ending with tags $u, v$ at position $j$
- so,

$$\pi(j, u, v) = \max_{<t_1, t_2, ...t_{j-2}>} \text{score}(t_1, t_2, ...t_{j-2}, u, v)$$

- Recursively:
  base with $\pi(0, \text{START}, \text{START}) = 0$
  for any $j \in 1, 2, ..., n$, for possible $u$ and $v$:

$$\pi(j, u, v) = \max_t (\pi(j-1, t, u) + \sum_i \lambda_i f'_i(word_v, t, u, v))$$

- the Viterbi Algorithm with Backpointers $\rightarrow$ the optimal sequence!

## More about Perceptron

- Voted Perceptron (Collins 2002)
- Averaged Perceptron (Collins 2002)
- Early Update (Collins and Roak 2004)

Questions

- can this model take features like:
  how many times we see a verb in this sentence ?

## Readings

**1999** Large Margin Classification using the Perceptron Algorithm, Machine Learning, 1999

**2002** Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. Michael Collins, EMNLP, 2002