Discriminative Sequence Models

Yoav Goldberg

Bar Ilan University

Generic NLP Solution

- Find an annotated corpus
- Split it into train and test parts
- Convert it to a vector representation
  - Decide on output type
  - Decide on features
  - Convert each training example to feature vector
- Train a machine-learning model on training set
- Apply machine-learning model to test set
- Measure accuracy

Reminder

- Feature function $\phi(x)$ or $\phi(x, y)$.
- Linear classifier $predict(x) = argmax_y \mathbf{w}\phi(x, y) + \mathbf{b}$

Setting weights: the multiclass perceptron algorithm

$\mathbf{w} \leftarrow 0$
for x, y in training examples **do**
    $\hat{y} \leftarrow \arg\max_{y'}(\mathbf{w} \cdot \phi(x, y'))$
    if $\hat{y} \neq y$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \phi(x, y)$
        $\mathbf{w} \leftarrow \mathbf{w} - \phi(x, \hat{y})$
return $\mathbf{w}$

Setting weights: the multiclass perceptron algorithm

$\mathbf{w} \leftarrow 0$
for x, y in training examples **do**
    $\hat{y} \leftarrow \arg\max_{y'}(\mathbf{w} \cdot \phi(x, y'))$
    if $\hat{y} \neq y$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \phi(x, y) - \phi(x, \hat{y})$
return $\mathbf{w}$

Discriminative POS-tagging

POS-tagging

$$x = x_1, \ldots, x_n = \text{Holly came from Miami , FLA}$$

$$y = y_1, \ldots, y_n = \text{NNP VBD IN NNP , NNP}$$

POS-tagging

$$x = x_1, \ldots, x_n = \text{Holly came from Miami , FLA}$$

$$y = y_1, \ldots, y_n = \text{NNP VBD IN NNP , NNP}$$

- Our job is to predict y. We will score each y_i in turn.

POS-tagging

$$x = x_1, \ldots, x_n = \text{Holly came from Miami , FLA}$$

$$y = y_1, \ldots, y_n = \text{NNP VBD IN NNP , NNP}$$

- Our job is to predict y. We will score each y_i in turn.
  - Reminder: in the HMM case we had:

$$score(y_i|x, y_{i-1}, y_{i-2}) = q(y_i|y_{i-1}, y_{i-2})e(x_i|y_i)$$

where e, q are MLE estimates.

POS-tagging

$$x = x_1, \ldots, x_n = \text{Holly came from Miami , FLA}$$

$$y = y_1, \ldots, y_n = \text{NNP VBD IN NNP , NNP}$$

- Our job is to predict y. We will score each y_i in turn.
  - Reminder: in the HMM case we had:

$$score(y_i|x, y_{i-1}, y_{i-2}) = q(y_i|y_{i-1}, y_{i-2})e(x_i|y_i)$$

    where e, q are MLE estimates.
- In the feature based approach:

$$predict(y_i|x, y_{[0:i-1]}) = \arg\max_{y_i \in tagset} w \cdot \phi(x, y_{[0:i-1]}, i, y_i)$$

- We define ϕ to take position into account: $\phi(x, y_{[0:i-1]}, i, y_i)$

POS-tagging

Feature Examples

Sequence:

$$\phi_{349} = \begin{cases} 1 & \text{if } y_{i-1}\text{='JJ', } \wedge\ y_{i-2}\text{='DT' } \wedge\ y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

$$\phi_{355} = \begin{cases} 1 & \text{if } \wedge\ y_{i-1}\text{='JJ' } \wedge\ y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

$$\phi_{392} = \begin{cases} 1 & \text{if } y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

Local:

$$\phi_{231} = \begin{cases} 1 & \text{if } x_i\text{='saw' } \wedge\ y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

POS-tagging

Feature Examples

Sequence:

$$\phi_{349} = \begin{cases} 1 & \text{if } y_{i-1}\text{='JJ'}, \wedge\ y_{i-2}\text{='DT'} \wedge\ y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

$$\phi_{355} = \begin{cases} 1 & \text{if } \wedge\ y_{i-1}\text{='JJ'} \wedge\ y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

$$\phi_{392} = \begin{cases} 1 & \text{if } y_i\text{='NN'} \\ 0 & otherwise \end{cases}$$

Local:

$$\phi_{231} = \begin{cases} 1 & \text{if } x_i\text{='saw'} \wedge\ y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

- These correspond to q and e from the HMM.
- Did we gain anything yet?

POS-tagging

More Feature Examples

3 Suffix:

$$\phi_{121} = \begin{cases} 1 & \text{if } x_i \text{ endsWith 'ing'} \wedge y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

$$\phi_{122} = \begin{cases} 1 & \text{if } x_i \text{ endsWith 'int'} \wedge y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

$\ldots$

2 Suffix:

$$\phi_{222} = \begin{cases} 1 & \text{if } x_i \text{ endsWith 'ed'} \wedge y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

$$\phi_{225} = \begin{cases} 1 & \text{if } x_i \text{ endsWith 'ng'} \wedge y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

$$\phi_{228} = \begin{cases} 1 & \text{if } x_i \text{ endsWith 'he'} \wedge y_i\text{='VBD'} \\ 0 & otherwise \end{cases}$$

$\ldots$

POS-tagging

More Feature Examples

Word-features:

$$\phi_{552} = \begin{cases} 1 & \text{if hasHyphen}(x_i) \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{553} = \begin{cases} 1 & \text{if hasDigit}(x_i) \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{554} = \begin{cases} 1 & \text{if allDigits}(x_i) \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{555} = \begin{cases} 1 & \text{if isUpper}(x_i) \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{556} = \begin{cases} 1 & \text{if allUpper}(x_i) \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

...

POS-tagging

More Feature Examples

Next word:

$$\phi_{621} = \begin{cases} 1 & \text{if } x_{i+1} \text{ endsWith 'ing'} \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{649} = \begin{cases} 1 & \text{if } x_{i+1}\text{='yesterday'} \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases} \quad \ldots$$

Prev word:

$$\phi_{721} = \begin{cases} 1 & \text{if } x_{i-1} \text{ endsWith 'ing'} \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$$\phi_{749} = \begin{cases} 1 & \text{if } x_{i-1}\text{='yesterday'} \wedge y_i\text{='JJ'} \\ 0 & otherwise \end{cases}$$

$\ldots$

POS-tagging

- We gained much more flexibility in what information sources we can use in the scoring.
- But how do we tag?
  - Can we use viterbi?

POS-tagging

- We cannot use viterbi because the scores are "meaningless".
- But greedy tagging works quite well!

1: **Input:** sentence x, parameter-vector $\mathbf{w}$, feature extractor ϕ
2: $y \leftarrow None$
3: **for** i in $1, \ldots, |x|$ **do**
4: $\quad y_i \leftarrow \arg\max_{y_i \in tagset} \mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)$
5: **return** y

POS-tagging

- We cannot use viterbi because the scores are "meaningless".
- But greedy tagging works quite well!

Input: sentence x, parameter-vector $\mathbf{w}$, feature extractor ϕ
$y \leftarrow None$
for i in $1, \ldots, |x|$ **do**
    $y_i \leftarrow \arg\max_{y_i \in tagset} \mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)$
return y

- This was a bad idea before. What changed?

Improving Greedy Tagging

Training Algorithm

1: $\mathbf{w} \leftarrow 0$
2: **for** x, y in examples **do**
3: **for** i in $1, \ldots, |x|$ **do**
4: $\hat{y}_i \leftarrow \arg\max_{y_i \in tagset} \mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)$
5: **if** $\hat{y}_i \neq y_i$ **then**
6: $\mathbf{w} \leftarrow \mathbf{w} + \phi(x, y_{[0:i-1]}, i, y_i) - \phi(x, y_{[0:i-1]}, i, \hat{y}_i)$
7: **return** $\mathbf{w}$

POS-tagging

Improving Greedy Tagging

Better Training Algorithm

1: $\mathbf{w} \leftarrow 0$
2: **for** x, y in examples **do**
3: $\quad y' \leftarrow y$
4: $\quad$ **for** i in $1, \ldots, |x|$ **do**
5: $\quad\quad \hat{y}_i \leftarrow \arg\max_{y_i \in tagset} \mathbf{w} \cdot \phi(x, y'_{[0:i-1]}, i, y_i)$
6: $\quad\quad$ **if** $\hat{y}_i \neq y_i$ **then**
7: $\quad\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \phi(x, y'_{[0:i-1]}, i, y_i) - \phi(x, y'_{[0:i-1]}, i, \hat{y}_i)$
8: $\quad\quad y'_i \leftarrow \hat{y}_i$
9: **return** $\mathbf{w}$

POS-tagging

Improving Greedy Tagging

Better Training Algorithm

1: $\mathbf{w} \leftarrow 0$
2: **for** x, y in examples **do**
3: $\quad y' \leftarrow y$
4: $\quad$ **for** i in $1, \ldots, |x|$ **do**
5: $\quad\quad \hat{y}_i \leftarrow \arg\max_{y_i \in tagset} \mathbf{w} \cdot \phi(x, y'_{[0:i-1]}, i, y_i)$
6: $\quad\quad$ **if** $\hat{y}_i \neq y_i$ **then**
7: $\quad\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \phi(x, y'_{[0:i-1]}, i, y_i) - \phi(x, y'_{[0:i-1]}, i, \hat{y}_i)$
8: $\quad\quad y'_i \leftarrow \hat{y}_i$
9: **return** $\mathbf{w}$

- Predict y_i based on previous predictions, can recover from errors.

POS-tagging

- Greedy tagging can be quite accurate (and very fast).

POS-tagging

- Greedy tagging can be quite accurate (and very fast).
- But a global search is still preferable.
- What if we do want to have global search?

POS-tagging

- Greedy tagging can be quite accurate (and very fast).
- But a global search is still preferable.
- What if we do want to have global search?

$\Rightarrow$ We need meaningful scores.

Log-linear Modeling

Log-linear Modeling

- Our linear classifiers give us a score:

$$score(x, y_{[0:i-1]}, i, y_i) = \mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)$$

- This is great, if we are looking for the best y.
- But sometimes we do want a probability:

$$p(y_i | x, y_{[0:i-1]}, i)$$

Log-linear Modeling

- Our linear classifiers give us a score:

$$score(x, y_{[0:i-1]}, i, y_i) = \mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)$$

- This is great, if we are looking for the best y.
- But sometimes we do want a probability:

$$p(y_i | x, y_{[0:i-1]}, i)$$

- In a log linear model, we define:

$$p(y_i | x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y'_i)}}$$

Log-linear Modeling

Name

- In a log linear model, we define:

$$p(y_i|x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

- Why is it called log-linear?

$$\log p(y_i|x, y_{[0:i-1]}, i) = \mathbf{w}\cdot\phi(x, y_{[0:i-1]}, i, y_i) - \log \sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}$$

Log-linear Modeling

Training Objective

- In a log linear model, we define:

$$p(y_i|x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

- The exponentiation makes everything positive.
- The denominator is normalizing everything to sum to 1.
  - $\Rightarrow$ The result is a formal probability.
  - $\Rightarrow$ But is it the "real" probability?

Log-linear Modeling

Training Objective

- In a log linear model, we define:

$$p(y_i|x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

- The exponentiation makes everything positive.
- The denominator is normalizing everything to sum to 1.
  - $\Rightarrow$ The result is a formal probability.
  - $\Rightarrow$ But is it the "real" probability?
- The job of learning is to find $\mathbf{w}$ such to maximize:

$$\sum_{x,y \in data} \sum_{i \in 1,\ldots,|x|} \log p(y_i|x, y_{[0:i-1]}, i)$$

(maximize the log likelihood of the data)

Log-linear Modeling

Smoothing

- In a log linear model, we define:

$$p(y_i|x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

- The job of learning is to find $\mathbf{w}$ such to maximize:

$$\sum_{x,y \in data} \sum_{i \in 1,\ldots,|x|} \log p(y_i|x, y_{[0:i-1]}, i)$$

Log-linear Modeling

Smoothing

- In a log linear model, we define:

$$p(y_i|x, y_{[0:i-1]}, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

- The job of learning is to find $\mathbf{w}$ such to maximize:

$$\sum_{x,y \in data} \sum_{i \in 1,\ldots,|x|} \log p(y_i|x, y_{[0:i-1]}, i)$$

- We can make it better by adding a regularization term:

$$\sum_{x,y \in data} \sum_{i \in 1,\ldots,|x|} \log p(y_i|x, y_{[0:i-1]}, i) - \frac{\lambda}{2} \sum w_i^2$$

- This prefers that most weights are small $\rightarrow$ avoid overfitting

Log-linear Modeling

Note

- This form of the model and objective is specialized to our tagging setup:

$$p(y_i|x, y, i) = \frac{e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y_i)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y_{[0:i-1]},i,y'_i)}}$$

$$\arg\max_{\mathbf{w}\in\mathbb{R}^m} \sum_{x,y\in data} \sum_{i\in 1,\ldots,|x|} \log p(y_i|x, y_{[0:i-1]}, i) - \frac{\lambda}{2}\sum w_i^2$$

- The general form is:

$$p(y|x) = \frac{e^{\mathbf{w}\cdot\phi(x,y)}}{\sum_{y'} e^{\mathbf{w}\cdot\phi(x,y')}}$$

$$\arg\max_{\mathbf{w}\in\mathbb{R}^m} \sum_{x,y\in data} \log p(y|x) - \frac{\lambda}{2}\sum w_i^2$$

Log-linear Modeling

Summary

- We defined $p(y|x) = \frac{e^{\mathbf{w}\cdot\phi(x,y)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y)}}$
- And train the model to optimize the data log-likelihood.
  - We also added a regularization term.

Log-linear Modeling

Summary

- We defined $p(y|x) = \frac{e^{\mathbf{w}\cdot\phi(x,y)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y)}}$
- And train the model to optimize the data log-likelihood.
  - We also added a regularization term.
- This is a very common approach, and goes by two names:
  - Multinomail Logistic Regression
  - Maximum Entropy Model (Maxent)

Log-linear Modeling

Summary

- We defined $p(y|x) = \frac{e^{\mathbf{w}\cdot\phi(x,y)}}{\sum_{y'_i \in tagset} e^{\mathbf{w}\cdot\phi(x,y)}}$
- And train the model to optimize the data log-likelihood.
  - We also added a regularization term.
- This is a very common approach, and goes by two names:
  - Multinomail Logistic Regression
  - Maximum Entropy Model (Maxent)
- There are many tools for training such models:
  - MegaM
  - Weka
  - **Liblinear** (when you pass the correct options)
  - VW (when you pass in the correct options)
  - **scikit-learn** (when you pass in the correct options)
  - Many others
  - (some of you implemented it in the deep-learning course...)

Back to tagging

MEMM – Maximum Entropy Markov Model

MEMM – MaxEnt Markov Models

- In a trigram HMM, we had:

$$p(x, y) = \prod_{i \in 1,\ldots,|x|} q(y_i | y_{i-1}, y_{i-2}) e(x_i | y_i)$$

$$\hat{y} = \arg\max_t p(x, y)$$

MEMM – MaxEnt Markov Models

- In a trigram HMM, we had:

$$p(x, y) = \prod_{i \in 1,\ldots,|x|} q(y_i | y_{i-1}, y_{i-2}) e(x_i | y_i)$$

$$\hat{y} = \arg\max_t p(x, y)$$

- In a trigram MEMM we have:

$$p(y|x) = \prod_{i \in 1,\ldots,|x|} p(y_i | x, y_{i-1}, y_{i-2})$$

$$\hat{y} = \arg\max_y p(y|x)$$

MEMM – MaxEnt Markov Models

- In a trigram HMM, we had:

$$p(x, y) = \prod_{i \in 1,\ldots,|x|} q(y_i | y_{i-1}, y_{i-2}) e(x_i | y_i)$$

$$\hat{y} = \arg\max_t p(x, y)$$

- In a trigram MEMM we have:

$$p(y|x) = \prod_{i \in 1,\ldots,|x|} p(y_i | x, y_{i-1}, y_{i-2}) = \prod_{i \in 1,\ldots,|x|} \frac{e^{\mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y_i)}}{\sum_{y'_i} e^{\mathbf{w} \cdot \phi(x, y_{[0:i-1]}, i, y'_i)}}$$

$$\hat{y} = \arg\max_y p(y|x)$$

MEMM – MaxEnt Markov Models

- In a trigram MEMM we have:

$$p(y|x) = \prod_{i \in 1,\ldots,|x|} p(y_i|x, y_{i-1}, y_{i-2}) = \prod_{i \in 1,\ldots,|x|} \frac{e^{\mathbf{w}\cdot\phi(x, y_{[0:i-1]}, i, y_i)}}{\sum_{y'_i} e^{\mathbf{w}\cdot\phi(x, y_{[0:i-1]}, i, y'_i)}}$$

$$\hat{y} = \arg\max_y p(y|x)$$

$\Rightarrow$ Train $\mathbf{w}$ using a Maxent learner (new)

$\Rightarrow$ Decode (solve the argmax) using viterbi (like before)

POS-tagging

Summary

- MEMM combine the flexibility of the feature-based approach with the global search and probability score of HMM
- But greedy decoding is very fast and very competitive.
- MEMM still a “locally-trained” model.

Structured Perceptron

POS-tagging

- MEMM combine the flexibility of the feature-based approach with the global search and probability score of HMM
- But greedy decoding is very fast and very competitive.
- MEMM still a “locally-trained” model.

$\Rightarrow$ Can we have a globally trained model?

POS-tagging

- MEMM combine the flexibility of the feature-based approach with the global search and probability score of HMM
- But greedy decoding is very fast and very competitive.
- MEMM still a “locally-trained” model.

$\Rightarrow$ Can we have a globally trained model?

- Assume for now that we do not care about probability.

POS-tagging

$\Rightarrow$ Can we have a globally trained model?

- Assume for now that we do not care about probability.

- Define $score(x, y, i, y_i) = \mathbf{w} \cdot \phi(x, y, i, y_i)$
- Then the sequence score is $score(x, y) = \sum_i \mathbf{w} \cdot \phi(x, y, i, y_i)$
- We can find $\arg\max_y score(x, y)$ using viterbi.
  - (assuming ϕ is "well behaved" – not looking at y_{i-3} or y_{i+1})
- Let's write:

$$\Phi(x, y) = \sum_i \phi(x, y, i, y_i)$$

- and now: $score(x, y) = \mathbf{w} \cdot \Phi(x, y)$
- we need $\mathbf{w} \cdot \Phi(x, y) > \mathbf{w} \cdot \Phi(x, y')$ for all incorrect y'.

Structured Perceptron

$$\Phi(x, y) = \sum_i \phi(x, y, i, y_i)$$

$$score(x, y) = \mathbf{w} \cdot \Phi(x, y)$$

we need $\mathbf{w} \cdot \Phi(x, y) > \mathbf{w} \cdot \Phi(x, y')$ for all incorrect y'

Structured Perceptron

$$\Phi(x, y) = \sum_i \phi(x, y, i, y_i)$$

$$score(x, y) = \mathbf{w} \cdot \Phi(x, y)$$

we need $\mathbf{w} \cdot \Phi(x, y) > \mathbf{w} \cdot \Phi(x, y')$ for all incorrect y'

Training – structured perceptron

1: $\mathbf{w} \leftarrow 0$
2: **for** x, y in examples **do**
3: $\quad \hat{y} \leftarrow \arg\max_{y'} \mathbf{w} \cdot \Phi(x, y')$ ▷ Using viterbi
4: $\quad$ **if** $\hat{y} \neq y$ **then**
5: $\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \Phi(x, y) - \Phi(x, \hat{y})$
6: **return** $\mathbf{w}$

Structured Perceptron

$$\Phi(x, y) = \sum_i \phi(x, y, i, y_i)$$

$$score(x, y) = \mathbf{w} \cdot \Phi(x, y)$$

we need $\mathbf{w} \cdot \Phi(x, y) > \mathbf{w} \cdot \Phi(x, y')$ for all incorrect y'

Training – structured perceptron

1: $\mathbf{w} \leftarrow 0$
2: **for** x, y in examples **do**
3: $\quad \hat{y} \leftarrow \arg\max_{y'} \mathbf{w} \cdot \Phi(x, y')$ ▷ Using viterbi
4: $\quad$ **if** $\hat{y} \neq y$ **then**
5: $\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \Phi(x, y) - \Phi(x, \hat{y})$
6: **return** $\mathbf{w}$

Structured Perceptron Tagger

- Globally optimized – optimizing directly what we care about
- Does not provide probabilities

Structured Perceptron Tagger

- Globally optimized – optimizing directly what we care about
- Does not provide probabilities
  - There are global model that do provide probabilities – CRFs
  - CRFs are the log-linear version of the structured-percepton.

Summary

Discriminative Learning

- Feature representation, feature vectors
- Linear models (binary, multiclass)
  - The perceptron algorithm
- Log-linear models
  - Regularization

Summary

Discriminative Learning

- Feature representation, feature vectors
- Linear models (binary, multiclass)
  - The perceptron algorithm
- Log-linear models
  - Regularization

Sequence Tagging

- HMM (last time)
- Greedy, classifier based
- Greedy, classifier based – improved
- MEMM
- Structured Perceptron