Unit 3: Structured Learning

(part 2) Discriminative Models: Perceptron & CRF

Nov 2010

Liang Huang (lhuang@isi.edu)
Generative models

\[ \hat{t} = \arg \max_t P(t \mid w) \]
\[ = \arg \max_t \frac{P(w, t)}{P(w)} \]
\[ = \arg \max_t P(w, t) \]

- More work to learn to generate \( w \)
- But can handle incomplete data: \( P(w) = \sum_t P(w, t) \)
The generative story

\[ P(w, t) = P(t) \times P(w \mid t) \]

\[ \approx P(t_1) \prod_{i=2}^{n} P(t_i \mid t_{i-1}) P(\text{stop} \mid t_n) \times \prod_{i=1}^{n} P(w_i \mid t_i) \]

\[ P(t \mid w) \approx \prod_{i=1}^{n} P(t_i \mid w_i) \]

\[ P(t \mid w) \approx \prod_{i=1}^{n} P(t_i \mid w_i) \times P(t_1) \prod_{i=2}^{n} P(t_i \mid t_{i-1}) P(\text{stop} \mid t_n) \]

you generated \( t_i \) twice!
Perceptron is...

• an extremely simple algorithm
• almost universally applicable
• and works very well in practice
Perceptron is ...

- an extremely simple algorithm
- almost universally applicable
- and works very well in practice

vanilla perceptron  
(Rosenblatt, 1958)  

structured perceptron  
(Collins, 2002)  

the man bit the dog

DT NN VBD DT NN
Generic Perceptron

- online-learning: one example at a time
- learning by doing
  - find the best output under the current weights
  - update weights at mistakes

\[ x_i \rightarrow \text{inference} \rightarrow z_i \rightarrow y_i \rightarrow \text{update weights} \rightarrow w \rightarrow \text{inference} \]
Example: POS Tagging

- **gold-standard:** DT NN VBD DT NN
  - the man bit the dog

- **current output:** DT NN NN DT NN
  - the man bit the dog

- assume only two feature classes
  - tag bigrams
  - word/tag pairs

- weights ++: (NN,VBD) (VBD, DT) (VBD $\rightarrow$ bit)
- weights --: (NN, NN) (NN, DT) (NN $\rightarrow$ bit)
**Example: POS Tagging**

- **gold-standard:**
  - DT NN VBD DT NN y
  - the man bit the dog x

- **current output:**
  - DT NN NN DT NN z
  - the man bit the dog x

- **assume only two feature classes**
  - **tag bigrams**
  - **word/tag pairs**

- **weights ++:**
  - (NN, VBD)  
  - (VBD, DT)  
  - (VBD → bit)

- **weights --:**
  - (NN, NN)  
  - (NN, DT)  
  - (NN → bit)
**Example: POS Tagging**

- **gold-standard:**
  - DT NN VBD DT NN y
  - the man bit the dog x

- **current output:**
  - DT NN NN DT NN z
  - the man bit the dog x

- Assume only two feature classes
  - Tag bigrams
  - Word/tag pairs

  - **Weights ++:** (NN, VBD) (VBD, DT) (VBD → bit)
  - **Weights --:** (NN, NN) (NN, DT) (NN → bit)
Example: POS Tagging

- **gold-standard:**
  - The man bit the dog
  - Assume only two feature classes
    - Tag bigrams
      - \( t_{i-1} t_i \)
    - Word/tag pairs
      - \( w_i \)
  - Weights ++:
    - (NN, VBD) (VBD, DT) (VBD → bit)
  - Weights --:
    - (NN, NN) (NN, DT) (NN → bit)

\[ \Phi(x, y) \]
\[ \Phi(x, z) \]
Example: POS Tagging

- **gold-standard:**
  - the man bit the dog

- **current output:**
  - the man bit the dog

- Assume only two feature classes
  - **tag bigrams**
  - **word/tag pairs**

- **weights ++:**
  - (NN, VBD)  (VBD, DT)  (VBD \rightarrow \text{bit})

- **weights --:**
  - (NN, NN)  (NN, DT)  (NN \rightarrow \text{bit})
Structured Perceptron

Inputs: Training set \((x_i, y_i)\) for \(i = 1 \ldots n\)

Initialization: \(W = 0\)

Define: \(F(x) = \text{argmax}_{y \in \text{GEN}(x)} \Phi(x, y) \cdot W\)

Algorithm: For \(t = 1 \ldots T, i = 1 \ldots n\)

\[ z_i = F(x_i) \]

If \(z_i \neq y_i\)

\[ W \leftarrow W + \Phi(x_i, y_i) - \Phi(x_i, z_i) \]

Output: Parameters \(W\)
Efficiency vs. Expressiveness

- the inference (argmax) must be efficient
  - either the search space $\text{GEN}(x)$ is small, or factored
  - features must be local to $y$ (but can be global to $x$)
    - e.g. bigram tagger, but look at all input words (cf. CRFs)
Efficiency vs. Expressiveness

- the inference (argmax) must be efficient
  - either the search space $\text{GEN}(x)$ is small, or factored
  - features must be local to $y$ (but can be global to $x$)
    - e.g. bigram tagger, but look at all input words (cf. CRFs)
Efficiency vs. Expressiveness

- The inference (argmax) must be efficient.
  - Either the search space $\text{GEN}(x)$ is small, or factored.
  - Features must be local to $y$ (but can be global to $x$).
    - E.g. bigram tagger, but look at all input words (cf. CRFs).

Discriminative Models
Efficiency vs. Expressiveness

- The inference \( \text{argmax}_{y \in \text{GEN}(x)} \) must be efficient
  - Either the search space \( \text{GEN}(x) \) is small, or factored
  - Features must be local to \( y \) (but can be global to \( x \))
    - E.g. bigram tagger, but look at all input words (cf. CRFs)
Efficiency vs. Expressiveness

- The inference (argmax) must be efficient.
  - Either the search space $\text{GEN}(x)$ is small, or factored.
  - Features must be local to $y$ (but can be global to $x$).
    - E.g. bigram tagger, but look at all input words (cf. CRFs).
Averaged Perceptron

Inputs: Training set $({x_i, y_i})$ for $i = 1 \ldots n$

Initialization: $W_0 = 0$

Define: $F(x) = \arg \max_{y \in \text{GEN}(x)} \Phi(x, y) \cdot W$

Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $W_{j+1} \leftarrow W_j + \Phi(x_i, y_i) - \Phi(x_i, z_i)$

Output: Parameters $W = \sum_j W_j$

- more stable and accurate results
- approximation of voted perceptron (Freund & Schapire, 1999)
Averaging Tricks

- Daume (2006, PhD thesis)

Algorithm AVERAGED STRUCTURED PERCEPTRON($x_{1:N}, y_{1:N}, I$)

1: $w_0 \leftarrow \langle 0, \ldots, 0 \rangle$
2: $w_a \leftarrow \langle 0, \ldots, 0 \rangle$
3: $c \leftarrow 1$
4: for $i = 1 \ldots I$ do
5:   for $n = 1 \ldots N$ do
6:     $\hat{y}_n \leftarrow \text{arg max}_{y \in Y} w_0^\top \Phi(x_n, y_n)$
7:     if $y_n \neq \hat{y}_n$ then
8:       $w_0 \leftarrow w_0 + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)$
9:       $w_a \leftarrow w_a + c\Phi(x_n, y_n) - c\Phi(x_n, \hat{y}_n)$
10:  end if
11: $c \leftarrow c + 1$
12: end for
13: end for
14: return $w_0 - w_a/c$

Figure 2.3: The averaged structured perceptron learning algorithm.
Do we need smoothing?

- smoothing is much easier in discriminative models
- just make sure for each feature template, its subset templates are also included
  - e.g., to include \((t_0 \, \omega_0 \, \omega_{-1})\) you must also include
  - \((t_0 \, \omega_0) \, (t_0 \, \omega_{-1}) \, (\omega_0 \, \omega_{-1})\)
  - and maybe also \((t_0 \, t_{-1})\) because \(t\) is less sparse than \(\omega\)
Comparison with Other Models
from HMM to MEMM

HMM: joint distribution

MEMM: locally normalized (per-state conditional)

Thursday, November 18, 2010
Label Bias Problem

- bias towards states with fewer outgoing transitions
- a problem with all locally normalized models
Conditional Random Fields

- globally normalized (no label bias problem)
- but training requires expected features counts
  - (related to the fractional counts in EM)
- need to use Inside-Outside algorithm (sum)
- Perceptron just needs Viterbi (max)
Experiments
Experiments: Tagging

• (almost) identical features from (Ratnaparkhi, 1996)
  • trigram tagger: current tag $t_i$, previous tags $t_{i-1}, t_{i-2}$
  • current word $w_i$ and its spelling features
  • surrounding words $w_{i-1} w_{i+1} w_{i-2} w_{i+2}$...

<table>
<thead>
<tr>
<th>Method</th>
<th>Error rate/%</th>
<th>Numits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perc, avg, cc=0</td>
<td>2.93</td>
<td>10</td>
</tr>
<tr>
<td>Perc, noavg, cc=0</td>
<td>3.68</td>
<td>20</td>
</tr>
<tr>
<td>Perc, avg, cc=5</td>
<td>3.03</td>
<td>6</td>
</tr>
<tr>
<td>Perc, noavg, cc=5</td>
<td>4.04</td>
<td>17</td>
</tr>
<tr>
<td>ME, cc=0</td>
<td>3.4</td>
<td>100</td>
</tr>
<tr>
<td>ME, cc=5</td>
<td><strong>3.28</strong></td>
<td>200</td>
</tr>
</tbody>
</table>
Experiments: NP Chunking

- **B-I-O** scheme

  Rockwell International Corp. B I I

  's Tulsa unit said it signed B I I O B O

  a tentative agreement ...

- **features:**
  - unigram model
  - surrounding words and POS tags

```
<table>
<thead>
<tr>
<th>Current word</th>
<th>wi</th>
<th>&amp; ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td>wi-1</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Word two back</td>
<td>wi-2</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Next word</td>
<td>wi+1</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Word two ahead</td>
<td>wi+2</td>
<td>&amp; ti</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bigram features</th>
<th>wi-2, wi-1 &amp; ti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wi-1, wi       &amp; ti</td>
</tr>
<tr>
<td></td>
<td>wi, wi+1       &amp; ti</td>
</tr>
<tr>
<td></td>
<td>wi+1, wi+2     &amp; ti</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current tag</th>
<th>pi</th>
<th>&amp; ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous tag</td>
<td>pi-1</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Tag two back</td>
<td>pi-2</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Next tag</td>
<td>pi+1</td>
<td>&amp; ti</td>
</tr>
<tr>
<td>Tag two ahead</td>
<td>pi+2</td>
<td>&amp; ti</td>
</tr>
</tbody>
</table>

| Bigram tag features | pi-2, pi-1 | & ti |
|                     | pi-1, pi   | & ti |
|                     | pi, pi+1   | & ti |
|                     | pi+1, pi+2 | & ti |

| Trigram tag features | pi-2, pi-1, pi | & ti |
|                      | pi-1, pi, pi+1 | & ti |
|                      | pi, pi+1, pi+2 | & ti |
```
Experiments: NP Chunking

- results

<table>
<thead>
<tr>
<th>Method</th>
<th>F-Measure</th>
<th>Numits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron, avg, cc=0</td>
<td>93.53</td>
<td>13</td>
</tr>
<tr>
<td>Perceptron, noavg, cc=0</td>
<td>93.04</td>
<td>35</td>
</tr>
<tr>
<td>Perceptron, avg, cc=5</td>
<td>93.33</td>
<td>9</td>
</tr>
<tr>
<td>Perceptron, noavg, cc=5</td>
<td>91.88</td>
<td>39</td>
</tr>
<tr>
<td>Max-ent, cc=0</td>
<td>92.34</td>
<td>900</td>
</tr>
<tr>
<td>Max-ent, cc=5</td>
<td>92.65</td>
<td>200</td>
</tr>
</tbody>
</table>

- (Sha and Pereira, 2003) trigram tagger
  - voted perceptron: 94.09% vs. CRF: 94.38%
Other NLP Applications

- dependency parsing (McDonald et al., 2005)
- parse reranking (Collins)
- phrase-based translation (Liang et al., 2006)
- word segmentation
- ... and many many more ...
Theory
Vanilla Perceptron
Vanilla Perceptron
Vanilla Perceptron
Convergence Theorem

- Data is separable if and only if perceptron converges
- number of updates is bounded by \((R/\gamma)^2\)
- \(\gamma\) is the margin; \(R = \max_i \|x_i\|\)
- This result generalizes to structured perceptron
  \[
  R = \max_i \|\Phi(x_i, y_i) - \Phi(x_i, z_i)\|
  \]
- Also in Collins paper: theorems for non-separable cases and generalization bounds
Perceptron Conclusion

- A very simple framework that can work with many structured problems and that works very well.
  - All you need is (fast) 1-best inference.
  - Much simpler than CRFs and SVMs.
  - Can be applied to parsing, translation, etc.

- Generalization bounds depend on separability.
  - Not the (exponential) size of the search space.

- Limitation 1: Features local on y (non-local on x).

- Limitation 2: No probabilistic interpretation (vs. CRF).
From HMM to CRF

(Sutton and McCallum, 2007) -- recommended reading

\[ p(y|x) = \frac{p(y, x)}{\sum_{y'} p(y', x)} = \frac{\exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t) \right\}}{\sum_{y'} \exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_{t-1}, x_t) \right\}}. \]
Maximum likelihood

\[ L = \log \prod_{j=1}^{N} P(t^j \mid w^j) \]

\[ = \log \prod_{j=1}^{N} \frac{1}{Z(w^j)} \text{exp} \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) \right) \]

\[ = \log \prod_{j=1}^{N} \frac{\exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) \right)}{\sum_t \exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t) \right)} \]

\[ = \sum_{j=1}^{N} \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) - \log \sum_t \exp \sum_{i=1}^{n} \lambda_i f_i(w^j, t) \right) \]
Maximum likelihood

\[ L = \log \prod_{j=1}^{N} P(t^j \mid w^j) \]

\[ = \log \prod_{j=1}^{N} \frac{1}{Z(w^j)} \exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) \right) \]

\[ = \log \prod_{j=1}^{N} \frac{\exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) \right)}{\sum_{t} \exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t) \right)} \]

\[ = \sum_{j=1}^{N} \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t^j) - \log \sum_{t} \exp \left( \sum_{i=1}^{n} \lambda_i f_i(w^j, t) \right) \right) \]
Optimizing likelihood

\[
\frac{\partial L}{\partial \lambda_i} = \sum_{j=1}^{N} \left( f_i(w^j, t^j) - E[f_i(w^j, t)] \right)
\]

- Gradient ascent:

\[
\lambda_i \leftarrow \lambda_i + \alpha \sum_{j=1}^{N} \left( f_i(w^j, t^j) - E[f_i(w^j, t)] \right)
\]

- Stochastic gradient ascent: for each \((w^j, t^j)\)

\[
\lambda_i \leftarrow \lambda_i + \alpha \left( f_i(w^j, t^j) - E[f_i(w^j, t)] \right)
\]
Expected feature counts

\[ E[f_{V,\text{loves}}] = \sum \frac{1}{Z} \alpha[F] \mu(V, \text{loves}) \beta[G] \]

\[ Z = \alpha[\text{END}] \]
CRF : SGD : Perceptron

- CRF = conditional random fields = structured maxent
- SGD = stochastic gradient descent = online CRF
- CRF => (online) => SGD => (viterbi) => Perceptron
- update rules (very similar):
  - CRF: batch, $E[...]
    \[ \lambda_i \leftarrow \lambda_i + \alpha \sum_{j=1}^{N} (f_i(w^j, t^j) - E[f_i(w^j, t)]) \]
  - SGD: one example at a time, expectation $E[...]
    \[ \lambda_i \leftarrow \lambda_i + \alpha \left( f_i(w^j, t^j) - E[f_i(w^j, t)] \right) \]
  - Perceptron: one example at a time, and Viterbi
    \[ \lambda_i \leftarrow \lambda_i + \alpha (f_i(w^j, t^j) - f_i(w^j, \hat{t})) \]