CRF and Structured Perceptron

CS 585, Fall 2015 -- Oct. 6
Introduction to Natural Language Processing
http://people.cs.umass.edu/~brenocon/inlp2015/

Brendan O’Connor
• Viterbi exercise solution
• CRF & Structured Perceptrons

• Thursday: project discussion + midterm review
Log-linear models (NB, LogReg, HMM, CRF...)

- $x$: Text Data
- $y$: Proposed class or sequence
- $\theta$: Feature weights (model parameters)
- $f(x,y)$: Feature extractor, produces feature vector

\[
    p(y|x) = \frac{1}{Z} \exp \left( \theta^T f(x, y) \right)
\]

Decision rule: $\arg \max_{y^* \in \text{outputs}(x)} G(y^*)$

How to we evaluate for HMM/CRF? Viterbi!
Things to do with a log-linear model

\[ p(y|x) = \frac{1}{Z} \exp \left( \theta^T f(x, y) \right) \]

\[ G(y) \]

**decoding/prediction**

\[ \arg \max_{y^* \in \text{outputs}(x)} G(y^*) \]

**parameter learning**

**feature engineering**

(human-in-the-loop)

\[ f(x, y) \text{ (feature vector)} \]
\[ \text{Text Input} \]
\[ \text{Output} \]
\[ \theta \text{ (Feature weights)} \]

- **given**
- **given**
- **obtain**

- **given**
- **given**
- **obtain**

- **given**
- **given**
- **obtain**

- **given**
- **given**

- **fiddle with**
- **during experiments**

\[ \text{obtain in each experiment} \]
HMM as factor graph

\[ p(y, w) = \prod_{t} p(w_t | y_t) \ p(y_{t+1} | y_t) \]

\[
\log p(y, w) = \sum_{t} \log p(w_t | y_t) + \log p(y_t | y_{t-1})
\]

(Additive) Viterbi: \[ \arg \max_{y^* \in \text{outputs}(x)} G(y^*) \]

\[ G(y) \]

\[ \text{emission factor score} \]

\[ \text{transition factor score} \]
is there a terrible bug in sutton&mccallum? there’s no sum over t in these equations!

We can write (1.13) more compactly by introducing the concept of feature functions, just as we did for logistic regression in (1.7). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (1.13), there needs to be one feature $f_{ij}(y, y', x) = 1_{\{y=i\}} 1_{\{y'=j\}}$ for each transition $(i, j)$ and one feature $f_{io}(y, y', x) = 1_{\{y=i\}} 1_{\{x=o\}}$ for each state-observation pair $(i, o)$. Then we can write an HMM as:

$$p(y, x) = \frac{1}{Z} \exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t) \right\}.$$  

(1.14)

**Definition 1.1**

Let $Y, X$ be random vectors, $\Lambda = \{\lambda_k\} \in \mathbb{R}^K$ be a parameter vector, and \{f_k(y, y', x_t)\}_{k=1}^{K} be a set of real-valued feature functions. Then a linear-chain conditional random field is a distribution $p(y|x)$ that takes the form

$$p(y|x) = \frac{1}{Z(x)} \exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t) \right\},$$

(1.16)
HMM as log-linear

\[
p(y, w) = \prod_{t} p(w_{y_t} | y_t) \ p(y_{t+1} | y_t)
\]

\[
\log p(y, w) = \sum_{t} \log p(w_t | y_t) + \log p(y_t | y_{t-1})
\]

\[
G(y) = \sum_{t} \left[ \sum_{k \in K} \sum_{w \in V} \mu_{w,t}1 \{y_t = k \land w_t = w\} + \sum_{k, j \in K} \lambda_{j,t}1 \{y_t = j \land y_{t+1} = k\} \right]
\]

\[
= \sum_{t} \sum_{i \in \text{allfeats}} \theta_{i,t}1 \{y_t, y_{t+1}, w_t\}
\]

\[
= \sum_{i \in \text{allfeats}} \theta_{i}1 \{y_t, y_{t+1}, w_t\}
\]

\[
\text{equation } 1.13, 1.14
\]

\[
\text{SM eq 1.13, 1.14}
\]
CRF

\[
\log p(y|x) = C + \theta^T f(x, y)
\]

\[
f(x, y) = \sum_t f_t(x, y_t, y_{t+1})
\]

- advantages
  - 1. why just word identity features? add many more!
  - 2. can train it to optimize accuracy of sequences (discriminative learning)
- Viterbi can be used for efficient prediction

Prob. dist over *whole* sequence

Linear-chain CRF: whole-sequence feature function decomposes into pairs

Tuesday, October 6, 15
Two simple feature templates

“Transition features”

\[ f_{\text{trans}}:A,B(x, y) = \sum_{t} 1\{y_{t-1} = A, y_{t} = B\} \]

“Observation features”

\[ f_{\text{emit}}:A,w(x, y) = \sum_{t} 1\{y_{t} = A, x_{t} = w\} \]

\( \text{gold} \ y = V \ V \ A \)}
Gold \( y = V \) \( A \)

**Transition features**

| -0.6 | -1.0 | 1.1 | 0.5 | 0.0 | 0.8 | 0.5 | -1.3 | -1.6 | 0.0 | 0.6 | 0.0 | -0.2 | -0.2 | 0.8 | -1.0 | 0.1 | -1.9 | 1.1 | 1.2 | -0.1 | -1.0 | -0.1 | -0.1 |

**Observation features**

| 1 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |

\[ f_{\text{trans}:V,A}(x, y) = \sum_{t=2}^{N} 1\{y_{t-1} = V, y_t = A\} \]

\[ f_{\text{obs}:V,\text{finna}}(x, y) = \sum_{t=1}^{N} 1\{y_t = V, x_t = \text{finna}\} \]

Goodness\( (y) = \theta^T f(x, y) \)

Mathematical convention is numeric indexing, though sometimes convenient to implement as hash table.
CRF: prediction with Viterbi

\[
\log p(y|x) = C + \theta^T f(x, y)
\]

\[
f(x, y) = \sum_t f_t(x, y_t, y_{t+1})
\]

- Scoring function has local decomposition

\[
f(x, y) = \sum_t f^{(B)}(t, x, y) + \sum_{t=2}^{T} f^{(A)}(y_{t-1}, y_t)
\]

\[
\theta^T f(x, y) = \sum_t \theta^T f^{(B)}(t, x, y) + \sum_{t=2}^{T} + f^{(A)}(y_{t-1}, y_t)
\]

Prob. dist over whole sequence

Linear-chain CRF: whole-sequence feature function decomposes into pairs
1. Motivation: we want features in our sequence model!

2. And how do we learn the parameters?

3. Outline

   1. Log-linear models
   2. Log-linear Sequence Models:
      1. Log-scale additive Viterbi
      2. Conditional Random Fields

3. Learning: the Perceptron
The Perceptron Algorithm

- Perceptron is not a model: it is a learning algorithm
  - Rosenblatt 1957

- Insanely simple algorithm
  - Iterate through dataset.
    - Predict.
    - Update weights to fix prediction errors.

- Can be used for classification OR structured prediction
  - *structured perceptron*

- Discriminative learning algorithm for *any* log-linear model (our view in this course)

The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights.
Binary perceptron

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT
      \[ y^* = \begin{cases} 
      \text{POS} & \text{if } \theta^T x \geq 0 \\
      \text{NEG} & \text{if } \theta^T x < 0 
      \end{cases} \]
  - IF y=y*, do nothing
  - ELSE update weights
    \[
    \theta := \theta + r x \\
    \theta := \theta - r x
    \]
    if POS misclassified as NEG: let’s make it more positive-y next time around
    if NEG misclassified as POS: let’s make it more negative-y next time

learning rate constant
e.g. r=1
Structured/multiclass Perceptron

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT
      \[ y^* = \arg \max_{y'} \theta^T f(x, y') \]
    - IF y=y*, do nothing
    - ELSE update weights
      \[ \theta := \theta + r [f(x, y) - f(x, y^*)] \]

  learning rate constant e.g. \( r=1 \)

  Features for TRUE label
  Features for PREDICTED label
Update rule

y=POS
x=“this awesome movie ...”
Make mistake: y*=NEG

\[
\theta := \theta + r [f(x, y) - f(x, y^*)]
\]

Learning rate e.g. \( r = 1 \)

Features for TRUE label

Features for PREDICTED label

<table>
<thead>
<tr>
<th></th>
<th>POS_awesome</th>
<th>POS_this</th>
<th>POS_oof</th>
<th>...</th>
<th>NEG_awesome</th>
<th>NEG_this</th>
<th>NEG_oof</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>pred</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>( f(x, \text{POS}) - f(x, \text{NEG}) = )</td>
<td>+1</td>
<td>+1</td>
<td>0</td>
<td>...</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>
For each feature $j$ in true $y$ but not predicted $y^*$:

$$\theta_j := \theta_j + (r) f_j(x, y)$$

For each feature $j$ not in true $y$, but in predicted $y^*$:

$$\theta_j := \theta_j - (r) f_j(x, y)$$
Two simple feature templates

“Transition features”

\[ f_{\text{trans}: A, B}(x, y) = \sum_t 1\{y_{t-1} = A, y_t = B\} \]

“Observation features”

\[ f_{\text{emit}: A, w}(x, y) = \sum_t 1\{y_t = A, x_t = w\} \]
Goodness(y) = $\theta^T f(x, y)$

$f_{\text{trans:} V, A}(x, y) = \sum_{t=2}^{N} 1\{y_{t-1} = V, y_t = A\}$

$f_{\text{obs:} V, \text{finna}}(x, y) = \sum_{t=1}^{N} 1\{y_t = V, x_t = \text{finna}\}$
Learning idea: want gold $y$ to have high scores. Update weights so $y$ would have a higher score, and $y^*$ would be lower, next time.

Perceptron update rule:

$$\theta := \theta + r[f(x, y) - f(x, y^*)]$$
\[ \theta := \theta + r[f(x, y) - f(x, y^*)] \]

The update vector:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0
\end{pmatrix}
\]

+ \( r \)
Perceptron notes/issues

- Issue: does it converge? (generally no)
- Solution: the averaged perceptron
- Can you regularize it? No... just averaging...

- By the way, there’s also likelihood training out there (gradient ascent on the log-likelihood function: the traditional way to train a CRF)
- structperc is easier to implement/conceptualize and performs similarly in practice
Averaged perceptron

- To get stability for the perceptron: *Voted* perc or *Averaged* perc
- See HW2 writeup
- Averaging: For $t$’th example... average together vectors from every timestep

$$\bar{\theta}_t = \frac{1}{t} \sum_{t'=1}^{t} \theta_{t'}$$

- Efficiency?
  - Lazy update algorithm in HW