Sequential Data Modeling - Conditional Random Fields

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Prediction Problems

Given $x$, predict $y$
# Prediction Problems

**Given \( x \), predict \( y \)**

<table>
<thead>
<tr>
<th><strong>A book review</strong></th>
<th><strong>Is it positive?</strong></th>
<th><strong>Binary Prediction</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Oh, man I love this book!</td>
<td>yes, no</td>
<td>(2 choices)</td>
</tr>
<tr>
<td>This book is so boring...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>A tweet</strong></th>
<th><strong>Its language</strong></th>
<th><strong>Multi-class Prediction</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>On the way to the park!</td>
<td>English, Japanese</td>
<td>(several choices)</td>
</tr>
<tr>
<td>公園に行くなう！</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>A sentence</strong></th>
<th><strong>Its parts-of-speech</strong></th>
<th><strong>Structured Prediction</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>I read a book</td>
<td>DET, VBD, NN</td>
<td>(millions of choices)</td>
</tr>
<tr>
<td>I read a book</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logistic Regression
Example we will use:

- Given an introductory sentence from Wikipedia
- Predict whether the article is about a person

<table>
<thead>
<tr>
<th>Given</th>
<th>Predict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gonso was a Sanron sect priest (754-827) in the late Nara and early Heian periods.</td>
<td>Yes!</td>
</tr>
<tr>
<td>Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.</td>
<td>No!</td>
</tr>
</tbody>
</table>

- This is binary classification (of course!)
Review: Linear Prediction Model

- Each element that helps us predict is a feature:
  - contains “priest”
  - contains “(<#>-<#>)”
  - contains “site”
  - contains “Kyoto Prefecture”

- Each feature has a weight, positive if it indicates “yes”, and negative if it indicates “no”:
  - \( w_{\text{contains “priest”}} = 2 \)
  - \( w_{\text{contains “(<#>-<#>)”}} = 1 \)
  - \( w_{\text{contains “site”}} = -3 \)
  - \( w_{\text{contains “Kyoto Prefecture”}} = -1 \)

- For a new example, sum the weights:
  - Kuya (903-972) was a priest born in Kyoto Prefecture.
  - \( 2 + (-1) + 1 = 2 \)

- If the sum is at least 0: “yes”, otherwise: “no”
Review: Mathematical Formulation

\[ y = \text{sign}(w \cdot \varphi(x)) \]
\[ = \text{sign}\left(\sum_{i=1}^{I} w_i \cdot \varphi_i(x)\right) \]

- \( x \): the input
- \( \varphi(x) \): vector of feature functions \( \{\varphi_1(x), \varphi_2(x), \ldots, \varphi_I(x)\} \)
- \( w \): the weight vector \( \{w_1, w_2, \ldots, w_I\} \)
- \( y \): the prediction, +1 if “yes”, -1 if “no”
  - (\( \text{sign}(v) \) is +1 if \( v \geq 0 \), -1 otherwise)
Perceptron and Probabilities

- Sometimes we want the probability $P(y|x)$
- Estimating confidence in predictions
- Combining with other systems
- However, perceptron only gives us a prediction

$$y = \text{sign}(w \cdot \varphi(x))$$

In other words:

$$P(y = 1|x) = 1 \text{ if } w \cdot \varphi(x) \geq 0$$
$$P(y = 1|x) = 0 \text{ if } w \cdot \varphi(x) < 0$$
The Logistic Function

- The logistic function is a “softened” version of the function used in the perceptron

\[ P(y = 1 | x) = \frac{e^{w \cdot \phi(x)}}{1 + e^{w \cdot \phi(x)}} \]

- Can account for uncertainty
- Differentiable
Logistic Regression

• Train based on **conditional likelihood**

• Find the parameters **w** that maximize the conditional likelihood of all answers **y_i** given the example **x_i**

\[ \hat{w} = \arg\max_w \prod_i P(y_i|x_i; w) \]

• How do we solve this?
Review: Perceptron Training Algorithm

```python
create map w
for / iterations
    for each labeled pair x, y in the data
        phi = CREATE_FEATURES(x)
        y' = PREDICT_ONE(w, phi)
        if y' != y
            w += y * phi
```

- In other words
  - Try to classify each training example
  - Every time we make a mistake, update the weights
Stochastic Gradient Descent

- Online training algorithm for probabilistic models (including logistic regression)

```python
create map w
for / iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

- In other words
  - For every training example, calculate the gradient (the direction that will increase the probability of y)
  - Move in that direction, multiplied by learning rate α
Gradient of the Logistic Function

- Take the derivative of the probability

\[
\frac{d}{dw} P(y = 1 | x) = \frac{d}{dw} \frac{e^{w \cdot \varphi(x)}}{1 + e^{w \cdot \varphi(x)}} = \varphi(x) \frac{e^{w \cdot \varphi(x)}}{(1 + e^{w \cdot \varphi(x)})^2}
\]

\[
\frac{d}{dw} P(y = -1 | x) = \frac{d}{dw} \left(1 - \frac{e^{w \cdot \varphi(x)}}{1 + e^{w \cdot \varphi(x)}}\right) = -\varphi(x) \frac{e^{w \cdot \varphi(x)}}{(1 + e^{w \cdot \varphi(x)})^2}
\]
Example: Initial Update

- Set $\alpha=1$, initialize $w=0$

$x =$ A site, located in Maizuru, Kyoto  \quad y = -1$

\[
\begin{align*}
\mathbf{w} \cdot \varphi(x) &= 0 \\
\frac{d}{dw} P(y=-1|x) &= -\frac{e^0}{(1+e^0)^2} \varphi(x) \\
&= -0.25 \varphi(x)
\end{align*}
\]

\[
\mathbf{w} \leftarrow \mathbf{w} + -0.25 \varphi(x)
\]

- $W_{\text{unigram "Maizuru"}} = -0.25$
- $W_{\text{unigram ","}} = -0.5$
- $W_{\text{unigram "in"}} = -0.25$
- $W_{\text{unigram "Kyoto"}} = -0.25$
- $W_{\text{unigram "A"}} = -0.25$
- $W_{\text{unigram "site"}} = -0.25$
- $W_{\text{unigram "located"}} = -0.25$
Example: Second Update

\( x = \text{Shoken, monk born in Kyoto} \)

\[ w \cdot \varphi(x) = -1 \]

\[ \frac{d}{dw} P(y = 1| x) = \frac{e^1}{(1+e^1)^2} \varphi(x) \]

\[ = 0.196 \varphi(x) \]

\[ w \leftarrow w + 0.196 \varphi(x) \]

\( y = 1 \)

\[ w \]

\( W_{\text{unigram "Maizuru"}} = -0.25 \]

\( W_{\text{unigram "A"}} = -0.25 \]

\( W_{\text{unigram "Shoken"}} = 0.196 \]

\( W_{\text{unigram "monk"}} = 0.196 \]

\( W_{\text{unigram "born"}} = 0.196 \]

\( W_{\text{unigram "Kyoto"}} = -0.25 \]

\( W_{\text{unigram "in"}} = -0.054 \]

\( W_{\text{unigram "located"}} = -0.25 \]

\( W_{\text{unigram "site"}} = -0.25 \]
Calculating Optimal Sequences, Probabilities
Sequence Likelihood

- Logistic regression considered probability of $y \in \{-1, +1\}$

$$P(y|x)$$

- What if we want to consider probability of a sequence?

$$P(Y|X)$$

Example sequence:

```
X_i → I → visited → Nara
Y_i → PRN → VBD → NNP
```
Calculating Multi-class Probabilities

- Each sequence has its own feature vector

\[ \phi_{T,<S>,N} = 1 \]
\[ \phi_{T,N,V} = 1 \]
\[ \phi_{T,V,<S>} = 1 \]
\[ \phi_{E,N,time} = 1 \]
\[ \phi_{E,V,flies} = 1 \]

- Use weights for each feature to calculate scores

\[ w_{T,<S>,N} = 1 \]
\[ w_{T,V,<S>} = 1 \]
\[ w_{E,N,time} = 1 \]

\[ \phi_{time\ flies}^{(N \ \ V)} \ast w = 3 \]
\[ \phi_{time\ flies}^{(V \ \ N)} \ast w = 0 \]
\[ \phi_{time\ flies}^{(N \ \ N)} \ast w = 2 \]
\[ \phi_{time\ flies}^{(V \ \ V)} \ast w = 1 \]
The Softmax Function

- Turn into probabilities by taking exponent and normalizing (the Softmax function)

\[
P(Y|X) = \frac{e^{w \cdot \varphi(Y, X)}}{\sum_{\tilde{Y}} e^{w \cdot \varphi(\tilde{Y}, X)}}
\]

- Take the exponent and normalize

\[
\begin{align*}
\exp(\varphi(\text{time flies})*w) &= 20.08 \\
\exp(\varphi(\text{time flies})*w) &= 1.00 \\
\exp(\varphi(\text{time flies})*w) &= 7.39 \\
\exp(\varphi(\text{time flies})*w) &= 2.72
\end{align*}
\]

\[
\begin{align*}
P(N V | \text{time flies}) &= 0.6437 \\
P(V N | \text{time flies}) &= 0.0320 \\
P(N N | \text{time flies}) &= 0.2369 \\
P(V V | \text{time flies}) &= 0.0872
\end{align*}
\]
Calculating Edge Features

- Like perceptron, can calculate features for each edge
Calculating Edge Probabilities

- Calculate scores, and take exponent

\[ e^{w \phi} = 7.39 \quad P = 0.881 \]
\[ e^{w \phi} = 1.00 \quad P = 0.237 \]
\[ e^{w \phi} = 1.00 \quad P = 0.119 \]

- This is now the same form as the HMM
  - Can use the Viterbi algorithm
  - Calculate probabilities using forward-backward
Conditional Random Fields
Maximizing CRF Likelihood

- Want to maximize the likelihood for sequences

\[ \hat{w} = \arg \max_w \prod_i P(Y_i | X_i ; w) \]

\[ P(Y | X) = \frac{e^{w \cdot \varphi(Y, X)}}{\sum_{\tilde{Y}} e^{w \cdot \varphi(\tilde{Y}, X)}} \]

- For convenience, we consider the log likelihood

\[ \log P(Y | X) = w \cdot \varphi(Y, X) - \log \sum_{\tilde{Y}} e^{w \cdot \varphi(\tilde{Y}, X)} \]

- Want to find gradient for stochastic gradient descent

\[ \frac{d}{dw} \log P(Y | X) \]
Deriving a CRF Gradient:

\[
\log P(Y|X) = w \cdot \varphi(Y, X) - \log \sum \tilde{Y} e^{w \cdot \varphi(\tilde{Y}, X)} \\
= w \cdot \varphi(Y, X) - \log Z
\]

\[
\frac{d}{dw} \log P(Y|X) = \varphi(Y, X) - \frac{d}{dw} \log \sum \tilde{Y} e^{w \cdot \varphi(\tilde{Y}, X)} \\
= \varphi(Y, X) - \frac{1}{Z} \sum \tilde{Y} \frac{d}{dw} e^{w \cdot \varphi(\tilde{Y}, X)} \\
= \varphi(Y, X) - \sum \tilde{Y} \frac{e^{w \cdot \varphi(\tilde{Y}, X)}}{Z} \varphi(\tilde{Y}, X) \\
= \varphi(Y, X) - \sum \tilde{Y} P(\tilde{Y}|X) \varphi(\tilde{Y}, X)
\]
In Other Words...

• To get the gradient we:

$$\frac{d}{d\mathbf{w}} \log P(\mathbf{Y}|\mathbf{X}) = \varphi(\mathbf{Y}, \mathbf{X}) - \sum \tilde{\mathbf{Y}} P(\tilde{\mathbf{Y}}|\mathbf{X}) \varphi(\tilde{\mathbf{Y}}, \mathbf{X})$$

add the correct feature vector

subtract the expectation of the features
Example

\[ \phi_{T,<S>,N} = 1 \]
\[ \phi_{T,N,V} = 1 \]
\[ \phi_{T,V,<S>} = 1 \]
\[ \phi_{E,N,time} = 1 \]
\[ \phi_{E,V,flies} = 1 \]

P = .644

\[ \phi_{T,<S>,V} = 1 \]
\[ \phi_{T,V,N} = 1 \]
\[ \phi_{T,N,<S>} = 1 \]
\[ \phi_{E,V,time} = 1 \]
\[ \phi_{E,N,flies} = 1 \]

P = .032

\[ \phi_{T,<S>,N} = 1 \]
\[ \phi_{T,N,N} = 1 \]
\[ \phi_{T,N,<S>} = 1 \]
\[ \phi_{E,N,time} = 1 \]
\[ \phi_{E,N,flies} = 1 \]

P = .237

\[ \phi_{T,<S>,V} = 1 \]
\[ \phi_{T,V,V} = 1 \]
\[ \phi_{T,V,<S>} = 1 \]
\[ \phi_{E,V,time} = 1 \]
\[ \phi_{E,V,flies} = 1 \]

P = .087

\[ \phi_{T,<S>,N}, \phi_{E,N,time} = 1-.644-.237 = .119 \]
\[ \phi_{T,N,V} = 1-.644 = .356 \]

\[ \phi_{T,<S>,V}, \phi_{E,V,time} = 0-.032-.087 = -.119 \]
\[ \phi_{T,V,N} = 0-.032 = -.032 \]

\[ \phi_{T,V,<S>}, \phi_{E,V,flies} = 1-.644-.087 = .269 \]
\[ \phi_{T,N,N} = 0-.237 = -.237 \]

\[ \phi_{T,N,<S>}, \phi_{E,V,flies} = 0-.032-.237 = -.269 \]
\[ \phi_{T,V,V} = 0-.087 = -.087 \]
Combinatorial Explosion

- **Problem!**: The number of hypotheses is exponential.

\[
\frac{d}{d \mathcal{W}} \log P(Y|X) = \varphi(Y, X) - \sum_{\tilde{Y}} P(\tilde{Y}|X) \varphi(\tilde{Y}, X)
\]

\[O(T^{|X|})\]

\[T = \text{number of tags}\]
Calculate Feature Expectations using Edge Probabilities!

- If we know the edge probabilities, just multiply them!

\[ e^{w\phi} = 7.39 \quad \text{P} = 0.881 \]
\[ \phi_{E,N,\text{time}} = 1 \quad \phi_{T,<S>,N} = 1 \]
\[ \phi_{E,V,\text{time}} = 1 \quad \phi_{T,<S>,V} = 1 \]

\[ e^{w\phi} = 1.00 \quad \text{P} = 0.119 \]

Same answer as when we explicitly expand all \( Y \):

\[ \phi_{T,<S>,N}, \phi_{E,N,\text{time}} = 1 - 0.881 = 0.119 \]
\[ \phi_{T,<S>,V}, \phi_{E,V,\text{time}} = 0 - 0.119 = -0.119 \]

\[ \phi_{T,<S>,N}, \phi_{E,N,\text{time}} = 1 - 0.644 - 0.237 = 0.119 \]
\[ \phi_{T,<S>,V}, \phi_{E,V,\text{time}} = 0 - 0.032 - 0.087 = -0.119 \]
CRF Training Procedure

- Can perform stochastic gradient descent, like logistic regression

```plaintext
create map w
for / iterations
  for each labeled pair X, Y in the data
    gradient = φ(Y,X)
    calculate e^{φ(y,x)*w} for each edge
    run forward-backward algorithm to get P(edge)
    for each edge
      gradient -= P(edge)*φ(edge)
      w += α * gradient
```

- Only major difference is gradient calculation
- Learning rate α
Learning Algorithms
Batch Learning

• **Online Learning:** Update after each example

  
  **Online Stochastic Gradient Descent**
  
  ```
  create map w
  for / iterations
    for each labeled pair x, y in the data
      w += \alpha \cdot \frac{dP(y|x)}{dw}
  ```

• **Batch Learning:** Update after all examples

  
  **Batch Stochastic Gradient Descent**
  
  ```
  create map w
  for / iterations
    for each labeled pair x, y in the data
      gradient += \alpha \cdot \frac{dP(y|x)}{dw}
    w += gradient
  ```
Batch Learning Algorithms: Newton/Quasi-Newton Methods

- **Newton-Raphson Method:**
  - Choose how far to update using the **second-order derivatives** (the **Hessian** matrix)
  - Faster convergence, but $|\mathbf{w}|^*|\mathbf{w}|$ time and memory

- **Limited Memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS):**
  - Guesses **second-order derivatives** from first-order
  - Most widely used?

- **More information:**
Online Learning vs. Batch Learning

- **Online:**
  - In general, simpler mathematical derivation
  - Often converges faster

- **Batch:**
  - More stable (does not change based on order)
  - Trivially parallelizable
Regularization
Cannot Distinguish Between Large and Small Classifiers

• For these examples:

```
-1  he saw a bird in the park
+1  he saw a robbery in the park
```

• Which classifier is better?

<table>
<thead>
<tr>
<th>Classifier 1</th>
<th>Classifier 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>he +3</td>
<td>bird -1</td>
</tr>
<tr>
<td>saw -5</td>
<td>robbery +1</td>
</tr>
<tr>
<td>a +0.5</td>
<td></td>
</tr>
<tr>
<td>bird -1</td>
<td></td>
</tr>
<tr>
<td>robbery +1</td>
<td></td>
</tr>
<tr>
<td>in +5</td>
<td></td>
</tr>
<tr>
<td>the -3</td>
<td></td>
</tr>
<tr>
<td>park -2</td>
<td></td>
</tr>
</tbody>
</table>
Cannot Distinguish Between Large and Small Classifiers

- For these examples:

-1 he saw a bird in the park
+1 he saw a robbery in the park

- Which classifier is better?

<table>
<thead>
<tr>
<th>Classifier 1</th>
<th>Classifier 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>he +3</td>
<td>bird -1</td>
</tr>
<tr>
<td>saw -5</td>
<td>robbery +1</td>
</tr>
<tr>
<td>a +0.5</td>
<td></td>
</tr>
<tr>
<td>bird -1</td>
<td></td>
</tr>
<tr>
<td>robbery +1</td>
<td></td>
</tr>
<tr>
<td>in +5</td>
<td></td>
</tr>
<tr>
<td>the -3</td>
<td></td>
</tr>
<tr>
<td>park -2</td>
<td></td>
</tr>
</tbody>
</table>

Probably classifier 2! It doesn't use irrelevant information.
**Regularization**

- A penalty on adding extra weights
- **L2 regularization:**
  - Big penalty on large weights, small penalty on small weights
  - **High accuracy**
- **L1 regularization:**
  - Uniform increase whether large or small
  - Will cause many weights to become zero → **small model**
Regularization in Logistic Regression/CRF

- To do so in logistic regression/CRF, we add the penalty to the log likelihood (for the whole corpus)

\[
\hat{w} = \arg\max_w (\prod_i P(Y_i|X_i; w)) - c \sum_{w \in w} w^2
\]

- \( c \) adjusts the strength of the regularization
  - smaller: more freedom to fit the data
  - larger: less freedom to fit the data, better generalization

- L1 also used, slightly more difficult to optimize
Conclusion
Conclusion

- **Logistic regression** is a probabilistic classifier
- **Conditional random fields** are probabilistic structured discriminative prediction models
- Can be trained using
  - **Online** stochastic gradient descent (like perceptron)
  - **Batch** learning using a method such as L-BFGS
- **Regularization** can help solve problems of overfitting
Thank You!