NLP Programming Tutorial 3 - The Perceptron Algorithm

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

Given $x$, predict $y$
Prediction Problems

Given $x$, predict $y$

- **A book review**
  
  Oh, man I love this book!
  
  This book is so boring...
  
  Is it positive?  
  
  yes
  
  no

- **A tweet**
  
  On the way to the park!
  
  公園にに行くよう！

  Its language
  
  English
  
  Japanese

- **A sentence**
  
  I read a book

  Its syntactic parse
  
  I read a book

Binary Prediction
(2 choices)

Multi-class Prediction
(several choices)

Structured Prediction
(millions of choices)
Example we will use:

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

<table>
<thead>
<tr>
<th><strong>Given</strong></th>
<th><strong>Predict</strong></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!)
Performing Prediction
How do We Predict?

Gonso was a Sanron sect priest (754 – 827) in the late Nara and early Heian periods.

Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.
How do We Predict?

Gonso was a Sanron sect priest (754 – 827) in the late Nara and early Heian periods.

Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.

Contains “priest” → probably person!
Contains “(<#>-<#>)” → probably person!
Contains “site” → probably not person!
Contains “Kyoto Prefecture” → probably not person!
Combining Pieces of Information

- Each element that helps us predict is a *feature*

  - contains “priest”
  - contains “site”
  - contains “(<#>-<#>)”
  - contains “Kyoto Prefecture”

- Each feature has a weight, *positive* if it indicates “yes”, and *negative* if it indicates “no”

  \[
  \begin{align*}
  w_{\text{contains “priest”}} &= 2 \\
  w_{\text{contains “site”}} &= -3 \\
  w_{\text{contains “(<#>-<#>)”}} &= 1 \\
  w_{\text{contains “Kyoto Prefecture”}} &= -1
  \end{align*}
  \]

- 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”
Let me Say that in Math!

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

- **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”  
  - (sign(v) is +1 if v >= 0, -1 otherwise)
Example Feature Functions: Unigram Features

- Equal to “number of times a particular word appears”

\[
\begin{align*}
\varphi_{\text{unigram } "A"}(x) &= 1 \\
\varphi_{\text{unigram } "site"}(x) &= 1 \\
\varphi_{\text{unigram } ","}(x) &= 2 \\
\varphi_{\text{unigram } "located"}(x) &= 1 \\
\varphi_{\text{unigram } "in"}(x) &= 1 \\
\varphi_{\text{unigram } "Maizuru"}(x) &= 1 \\
\varphi_{\text{unigram } "Kyoto"}(x) &= 1 \\
\varphi_{\text{unigram } "the"}(x) &= 0 \\
\varphi_{\text{unigram } "temple"}(x) &= 0 \\
\end{align*}
\]

The rest are all 0

- For convenience, we use feature names (\(\varphi_{\text{unigram } "A"}\)) instead of feature indexes (\(\varphi_1\))
Calculating the Weighted Sum

\[ x = \text{A site, located in Maizuru, Kyoto} \]

\[
\begin{align*}
\phi_{\text{unigram "A"}}(x) &= 1 \\
\phi_{\text{unigram "site"}}(x) &= 1 \\
\phi_{\text{unigram "located"}}(x) &= 1 \\
\phi_{\text{unigram "Maizuru"}}(x) &= 1 \\
\phi_{\text{unigram ","}}(x) &= 2 \\
\phi_{\text{unigram "in"}}(x) &= 1 \\
\phi_{\text{unigram "Kyoto"}}(x) &= 1 \\
\phi_{\text{unigram "priest"}}(x) &= 0 \\
\phi_{\text{unigram "black"}}(x) &= 0
\end{align*}
\]

\[
\begin{align*}
W_{\text{unigram "a"}} &= 0 \\
W_{\text{unigram "site"}} &= -3 \\
W_{\text{unigram "located"}} &= 0 \\
W_{\text{unigram "Maizuru"}} &= 0 \\
W_{\text{unigram ","}} &= 0 \\
W_{\text{unigram "in"}} &= 0 \\
W_{\text{unigram "Kyoto"}} &= 0 \\
W_{\text{unigram "priest"}} &= 2 \\
W_{\text{unigram "black"}} &= 0
\end{align*}
\]

\[
\begin{array}{cccc}
0 & -3 & + \\
0 & -3 & + \\
0 & 0 & + \\
0 & 0 & + \\
0 & 2 & + \\
0 & 0 & + \\
\cdots & \cdots & + \\
\end{array}
\]

\[
\sum W \phi = -3 \rightarrow \text{No!}
\]
Pseudo Code for Prediction

```
PREICT_ALL(model_file, input_file):
    load w from model_file  # so w[name] = w_name
    for each x in input_file
        phi = CREATE_FEATURES(x)  # so phi[name] = φ_name(x)
        y' = PREDICT_ONE(w, phi)  # calculate sign(w*φ(x))
    print y'
```
Pseudo Code for Predicting a Single Example

```
PREDICT_ONE(w, phi)
    score = 0
    for each name, value in phi        # score = w*φ(x)
        if name exists in w
            score += value * w[name]
    if score >= 0
        return 1
    else
        return -1
```
Pseudo Code for Feature Creation
(Example: Unigram Features)

`CREATE_FEATURES(x):
  create map phi
  split x into words
  for word in words
    phi[“UNI:”+word] += 1  # We add “UNI:” to indicate unigrams
  return phi

• You can **modify this function** to use other features!
  • Bigrams?
  • Other features?
Learning Weights Using the Perceptron Algorithm
Learning Weights

- Manually creating weights is hard
  - Many many possible useful features
  - Changing weights changes results in unexpected ways
- Instead, we can learn from labeled data

<table>
<thead>
<tr>
<th>$y$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FUJIWARA no Chikamori (year of birth and death unknown) was a samurai and poet who lived at the end of the Heian period.</td>
</tr>
<tr>
<td>1</td>
<td>Ryonen (1646 - October 29, 1711) was a Buddhist nun of the Obaku Sect who lived from the early Edo period to the mid-Edo period.</td>
</tr>
<tr>
<td>-1</td>
<td>A moat settlement is a village surrounded by a moat.</td>
</tr>
<tr>
<td>-1</td>
<td>Fushimi Momoyama Athletic Park is located in Momoyama-cho, Kyoto City, Kyoto Prefecture.</td>
</tr>
</tbody>
</table>
Online Learning

```plaintext
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
            UPDATE_WEIGHTS(w, phi, y)
```

- In other words
  - Try to classify each training example
  - Every time we make a mistake, update the weights
- Many different online learning algorithms
  - The most simple is the perceptron
Perceptron Weight Update

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

• In other words:
  • If \( y=1 \), increase the weights for features in \( \varphi(x) \)
    - Features for positive examples get a higher weight
  • If \( y=-1 \), decrease the weights for features in \( \varphi(x) \)
    - Features for negative examples get a lower weight
→ Every time we update, our predictions get better!

```
UPDATE_WEIGHTS(w, phi, y)
for name, value in phi:
w[name] += value * y
```
Example: Initial Update

- Initialize $\mathbf{w}=0$

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

$\mathbf{w} \cdot \varphi(x) = 0 \quad y' = \text{sign}(\mathbf{w} \cdot \varphi(x)) = 1$

$y' \neq y$

$\mathbf{w} \leftarrow \mathbf{w} + y \varphi(x)$

$W_{\text{unigram "Maizuru"}} = -1$
$W_{\text{unigram ","}} = -2$
$W_{\text{unigram "in"}} = -1$
$W_{\text{unigram "Kyoto"}} = -1$

$W_{\text{unigram "A"}} = -1$
$W_{\text{unigram "site"}} = -1$
$W_{\text{unigram "located"}} = -1$
Example: Second Update

\[ x = \text{Shoken, monk born in Kyoto} \quad y = 1 \]

\[ w \cdot \varphi(x) = -4 \quad y' = \text{sign}(w \cdot \varphi(x)) = -1 \]

\[ y' \neq y \]

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

\[ W_{\text{unigram "Maizuru"}} = -1 \]
\[ W_{\text{unigram ","}} = -1 \]
\[ W_{\text{unigram "in"}} = 0 \]
\[ W_{\text{unigram "Kyoto"}} = 0 \]
\[ W_{\text{unigram "A"}} = -1 \]
\[ W_{\text{unigram "site"}} = -1 \]
\[ W_{\text{unigram "located"}} = -1 \]
\[ W_{\text{unigram "Shoken"}} = 1 \]
\[ W_{\text{unigram "monk"}} = 1 \]
\[ W_{\text{unigram "born"}} = 1 \]
Exercise
Exercise (1)

- Write two programs
  - train-perceptron: Creates a perceptron model
  - test-perceptron: Reads a perceptron model and outputs one prediction per line
- Test train-perceptron
  - Input: test/03-train-input.txt
  - Answer: test/03-train-answer.txt
Exercise (2)

- **Train** a model on data-en/titles-en-train.labeled
- **Predict** the labels of data-en/titles-en-test.word
- **Grade** your answers and report next week
  - script/grade-prediction.py data-en/titles-en-test.labeled your_answer
- **Extra challenge:**
  - Find places where the model makes a mistake and analyze why
  - Devise new features that could increase accuracy
Thank You!