Machine Learning for NLP

Perceptron Learning

Joakim Nivre

Uppsala University
Department of Linguistics and Philology

Slides adapted from Ryan McDonald, Google Research
Linear Classifiers

- Classifiers covered so far:
  - Decision trees
  - Nearest neighbor

- Next three lectures: linear classifiers

- Statistics from Google Scholar (October 2009):
  - “Maximum Entropy” & “NLP” 2660 hits, 141 before 2000
  - “SVM” & “NLP” 2210 hits, 16 before 2000
  - “Perceptron” & “NLP”, 947 hits, 118 before 2000

- All are linear classifiers and basic tools in NLP

- They are also the bridge to deep learning techniques
Outline

- **Today:**
  - Preliminaries: input/output, features, etc.
  - Perceptron
  - Assignment 2

- **Next time:**
  - Large-margin classifiers (SVMs, MIRA)
  - Logistic regression (Maximum Entropy)

- **Final session:**
  - Naive Bayes classifiers
  - Generative and discriminative models
Inputs and Outputs

- **Input:** \( x \in \mathcal{X} \)
  - e.g., document or sentence with some words \( x = w_1 \ldots w_n \), or a series of previous actions

- **Output:** \( y \in \mathcal{Y} \)
  - e.g., parse tree, document class, part-of-speech tags, word-sense

- **Input/output pair:** \( (x, y) \in \mathcal{X} \times \mathcal{Y} \)
  - e.g., a document \( x \) and its label \( y \)
  - Sometimes \( x \) is explicit in \( y \), e.g., a parse tree \( y \) will contain the sentence \( x \)
Feature Representations

- We assume a mapping from input-output pairs \((x, y)\) to a high dimensional feature vector
  \[ f(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m \]
- For some cases, i.e., binary classification \(\mathcal{Y} = \{-1, +1\}\), we can map only from the input to the feature space
  \[ f(x) : \mathcal{X} \rightarrow \mathbb{R}^m \]
- However, most problems in NLP require more than two classes, so we focus on the multi-class case
- For any vector \(v \in \mathbb{R}^m\), let \(v_j\) be the \(j^{th}\) value
Features and Classes

- All features must be **numerical**
  - Numerical features are represented directly as $f_i(x, y) \in \mathbb{R}$
  - Binary (boolean) features are represented as $f_i(x, y) \in \{0, 1\}$
- Multinomial (categorical) features must be **binarized**
  - Instead of: $f_i(x, y) \in \{v_0, \ldots, v_p\}$
  - We have: $f_{i+0}(x, y) \in \{0, 1\}, \ldots, f_{i+p}(x, y) \in \{0, 1\}$
  - Such that: $f_{i+j}(x, y) = 1$ iff $f_i(x, y) = v_j$
- We need distinct features for distinct output classes
  - Instead of: $f_i(x) \ (1 \leq i \leq m)$
  - We have: $f_{i+0m}(x, y), \ldots, f_{i+Nm}(x, y)$ for $\mathcal{Y} = \{0, \ldots, N\}$
  - Such that: $f_{i+jm}(x, y) = f_i(x)$ iff $y = y_j$
Examples

- \( x \) is a document and \( y \) is a label
  \[
  f_j(x, y) = \begin{cases} 
  1 & \text{if } x \text{ contains the word “interest”} \\
  & \text{and } y = \text{“financial”} \\
  0 & \text{otherwise}
  \end{cases}
  \]

  \( f_j(x, y) = \% \text{ of words in } x \text{ with punctuation and } y = \text{“scientific”} \)

- \( x \) is a word and \( y \) is a part-of-speech tag
  \[
  f_j(x, y) = \begin{cases} 
  1 & \text{if } x = \text{“bank” and } y = \text{Verb} \\
  0 & \text{otherwise}
  \end{cases}
  \]
Examples

- $x$ is a name, $y$ is a label classifying the name

\[
\begin{align*}
  f_0(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "George"} \\
    & \text{and } y = \text{"Person"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_1(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "Washington"} \\
    & \text{and } y = \text{"Person"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_2(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "Bridge"} \\
    & \text{and } y = \text{"Person"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_3(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "General"} \\
    & \text{and } y = \text{"Person"} \\
    0 & \text{otherwise}
  \end{cases}
\]

\[
\begin{align*}
  f_4(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "George"} \\
    & \text{and } y = \text{"Object"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_5(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "Washington"} \\
    & \text{and } y = \text{"Object"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_6(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "Bridge"} \\
    & \text{and } y = \text{"Object"} \\
    0 & \text{otherwise}
  \end{cases} \\
  f_7(x, y) &= \begin{cases} 
    1 & \text{if } x \text{ contains "General"} \\
    & \text{and } y = \text{"Object"} \\
    0 & \text{otherwise}
  \end{cases}
\]

- $x=$General George Washington, $y=$Person $\rightarrow f(x, y) = [1, 1, 0, 1, 0, 0, 0, 0]$
- $x=$George Washington Bridge, $y=$Object $\rightarrow f(x, y) = [0, 0, 0, 0, 1, 1, 1, 0]$
- $x=$George Washington George, $y=$Object $\rightarrow f(x, y) = [0, 0, 0, 0, 1, 1, 0, 0]$
Block Feature Vectors

- $x=$ General George Washington, $y=$ Person $\rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
- $x=$ George Washington Bridge, $y=$ Object $\rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$
- $x=$ George Washington George, $y=$ Object $\rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$

- One equal-size block of the feature vector for each label
- Input features duplicated in each block
- Non-zero values allowed only in one block
Linear Classifiers

- **Linear classifier**: score (or probability) of a particular classification is based on a linear combination of features and their weights.

- Let \( w \in \mathbb{R}^m \) be a high dimensional weight vector.

- If we assume that \( w \) is known, then we define our classifier as
  
  **Multiclass Classification**: \( Y = \{0, 1, \ldots, N\} \)
  
  \[
  y = \arg \max_y w \cdot f(x, y) = \arg \max_y \sum_{j=0}^{m} w_j \times f_j(x, y)
  \]

- **Binary Classification** just a special case of multiclass.
Linear Classifiers – Bias Terms

▶ Often linear classifiers presented as

\[ y = \arg \max_y \sum_{j=0}^{m} w_j \times f_j(x, y) + b_y \]

▶ Where \( b \) is a bias or offset term

▶ But this can be folded into \( f \)

\( x=\text{General George Washington}, \ y=\text{Person} \rightarrow f(x, y) = [1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \)

\( x=\text{General George Washington}, \ y=\text{Object} \rightarrow f(x, y) = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1] \)

\( f_4(x, y) = \begin{cases} 
1 & y = \text{“Person”} \\
0 & \text{otherwise}
\end{cases} \quad f_9(x, y) = \begin{cases} 
1 & y = \text{“Object”} \\
0 & \text{otherwise}
\end{cases} \)

▶ \( w_4 \) and \( w_9 \) are now the bias terms for the labels
Binary Linear Classifier

Divides all points:
Multiclass Linear Classifier

Defines regions of space:

- i.e., $+$ are all points $(x, y)$ where $+$ = $\arg \max_y w \cdot f(x, y)$
Separability

- A set of points is separable, if there exists a $w$ such that classification is perfect

- This can also be defined mathematically (and we will shortly)
Supervised Learning – how to find \( \mathbf{w} \)

- Input: training examples \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{|\mathcal{T}|} \)
- Input: feature representation \( \mathbf{f} \)
- Output: \( \mathbf{w} \) that maximizes/minimizes some important function on the training set
  - minimize error (Perceptron, SVMs, Boosting)
  - maximize likelihood of data (Logistic Regression, Naive Bayes)
- Assumption: The training data is separable
  - Not necessary, just makes life easier
  - There is a lot of good work in machine learning to tackle the non-separable case
Perceptron

Choose a $\mathbf{w}$ that minimizes error

$$
\mathbf{w} = \arg \min_{\mathbf{w}} \sum_{t} 1 - 1[y_t = \arg \max_y \mathbf{w} \cdot f(x_t, y)]
$$

$1[p] = \begin{cases} 
1 & p \text{ is true} \\
0 & \text{otherwise}
\end{cases}$

This is a 0-1 loss function

Aside: when minimizing error people tend to use hinge-loss or other smoother loss functions
Perceptron Learning Algorithm

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\lvert \mathcal{T} \rvert}$

1. $w^{(0)} = 0$; $i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y' = \arg \max_y w^{(i)} \cdot f(x_t, y)$
5. if $y' \neq y_t$
6. $w^{(i+1)} = w^{(i)} + f(x_t, y_t) - f(x_t, y')$
7. $i = i + 1$
8. return $w^i$
Perceptron: Separability and Margin

Given an training instance \((x_t, y_t)\), define:

\[ \bar{Y}_t = Y - \{y_t\} \]

i.e., \(\bar{Y}_t\) is the set of incorrect labels for \(x_t\)

A training set \(\mathcal{T}\) is separable with margin \(\gamma > 0\) if there exists a vector \(u\) with \(\|u\| = 1\) such that:

\[ u \cdot f(x_t, y_t) - u \cdot f(x_t, y') \geq \gamma \]

for all \(y' \in \bar{Y}_t\) and \(\|u\| = \sqrt{\sum_j u_j^2}\) (Euclidean or \(L^2\) norm)

Assumption: the training set is separable with margin \(\gamma\)
Perceptron: Main Theorem

Theorem: For any training set separable with a margin of $\gamma$, the following holds for the perceptron algorithm:

\[
\text{mistakes made during training} \leq \frac{R^2}{\gamma^2}
\]

where $R \geq ||f(x_t, y_t) - f(x_t, y')||$ for $(x_t, y_t) \in T$ and $y' \in \tilde{Y}_t$

Thus, after a finite number of training iterations, the error on the training set will converge to zero.

For proof, see Collins (2002)
Practical Considerations

- The perceptron is sensitive to the order of training examples
  - Consider: 500 positive instances + 500 negative instances
- Shuffling:
  - Randomly permute training instances between iterations
- Voting and averaging:
  - Let $\mathbf{w}_1, \ldots \mathbf{w}_n$ be all the weight vectors seen in training
  - The \textit{voted} perceptron predicts the majority vote of $\mathbf{w}_1, \ldots \mathbf{w}_n$
  - The \textit{averaged} perceptron predicts using the average vector:

$$\overline{\mathbf{w}} = \frac{1}{n} \sum_{i} \mathbf{w}_1, \ldots \mathbf{w}_n$$
Perceptron Summary

- Learns a linear classifier that minimizes error
  - Guaranteed to find a \( w \) in a finite amount of time
  - Improvement 1: shuffle training data between iterations
  - Improvement 2: average weight vectors seen during training
- Perceptron is an example of an online learning algorithm
  - \( w \) is updated based on a single training instance in isolation
    \[
    w^{(i+1)} = w^{(i)} + f(x_t, y_t) - f(x_t, y')
    \]
- Compare decision trees that perform batch learning
  - All training instances are used to find best split
Assignment 2

- Implement the perceptron (starter code in Python)
- Three subtasks:
  - Implement the linear classifier (dot product)
  - Implement the perceptron update
  - Evaluate on the spambase data set
- For VG
  - Implement the averaged perceptron
Appendix

Proofs and Derivations
Convergence Proof for Perceptron

Perceptron Learning Algorithm

Training data: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{||\mathcal{T}||} \)

1. \( w^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y' = \arg \max_y w^{(i)} \cdot f(x_t, y) \)
5. if \( y' \neq y_t \)
6. \( w^{(i+1)} = w^{(i)} + f(x_t, y_t) - f(x_t, y') \)
7. \( i = i + 1 \)
8. return \( w^i \)

- \( w^{(k-1)} \) are the weights before \( k^{th} \) mistake
- Suppose \( k^{th} \) mistake made at the \( t^{th} \) example, \((x_t, y_t)\)
- \( y' = \arg \max_y w^{(k-1)} \cdot f(x_t, y) \)
- \( y' \neq y_t \)
- \( w^{(k)} = w^{(k-1)} + f(x_t, y_t) - f(x_t, y') \)

- Now: \( u \cdot w^{(k)} = u \cdot w^{(k-1)} + u \cdot (f(x_t, y_t) - f(x_t, y')) \geq u \cdot w^{(k-1)} + \gamma \)
- Now: \( w^{(0)} = 0 \) and \( u \cdot w^{(0)} = 0 \), by induction on \( k \), \( u \cdot w^{(k)} \geq k \gamma \)
- Now: since \( u \cdot w^{(k)} \leq ||u|| \times ||w^{(k)}|| \) and \( ||u|| = 1 \) then \( ||w^{(k)}|| \geq k \gamma \)
- Now:

\[
||w^{(k)}||^2 = ||w^{(k-1)}||^2 + ||f(x_t, y_t) - f(x_t, y')||^2 + 2w^{(k-1)} \cdot (f(x_t, y_t) - f(x_t, y'))
\]

\[
||w^{(k)}||^2 \leq ||w^{(k-1)}||^2 + R^2
\]

(since \( R \geq ||f(x_t, y_t) - f(x_t, y')|| \))

and \( w^{(k-1)} \cdot f(x_t, y_t) - w^{(k-1)} \cdot f(x_t, y') \leq 0 \)
Perceptron Learning Algorithm

- We have just shown that $||w^{(k)}|| \geq k\gamma$ and $||w^{(k)}||^2 \leq ||w^{(k-1)}||^2 + R^2$

- By induction on $k$ and since $w^{(0)} = 0$ and $||w^{(0)}||^2 = 0$

  $||w^{(k)}||^2 \leq kR^2$

- Therefore,

  $k^2\gamma^2 \leq ||w^{(k)}||^2 \leq kR^2$

- and solving for $k$

  $k \leq \frac{R^2}{\gamma^2}$

- Therefore the number of errors is bounded!