A brief introduction to kernel classifiers

Mark Johnson
Brown University

October 2009
Outline

Introduction

Linear and nonlinear classifiers

Kernels and classifiers

The kernelized perceptron learner

Conclusions
Features and kernels are duals

- A **kernel** $K$ is a kind of similarity function
  - $K(x_1, x_2) > 0$ is the “similarity” of $x_1, x_2 \in \mathcal{X}$
- A **feature representation** $\mathbf{f}$ defines a kernel
  - $\mathbf{f}(x) = (f_1(x), \ldots, f_m(x))$ is feature vector
    \[
    K(x_1, x_2) = \mathbf{f}(x_1) \cdot \mathbf{f}(x_2) = \sum_{j=1}^{m} f_j(x_1) f_j(x_2)
    \]
- Mercer’s theorem: For every continuous symmetric positive semi-definite kernel $K$ there is a feature vector function $\mathbf{f}$ such that
  \[
  K(x_1, x_2) = \mathbf{f}(x_1) \cdot \mathbf{f}(x_2)
  \]
  - $\mathbf{f}$ may have **infinitely many dimensions**

$\Rightarrow$ Feature-based approaches and kernel-based approaches are often mathematically interchangable
  - Feature and kernel representations are **duals**
Learning algorithms and kernels

- Feature representations and kernel representations are duals
  ⇒ Many learning algorithms can use either features or kernels
    ▶ feature version maps examples into feature space and learns feature statistics
    ▶ kernel version uses “similarity” between this example and other examples, and learns example statistics
- Both versions *learn same classification function*
- Computational complexity of feature vs kernel algorithms can vary dramatically
  ▶ few features, many training examples
    ⇒ feature version may be more efficient
  ▶ few training examples, many features
    ⇒ kernel version may be more efficient
Outline

Introduction

Linear and nonlinear classifiers

Kernels and classifiers

The kernelized perceptron learner

Conclusions
Linear classifiers

- A **classifier** is a function $c$ that maps an example $x \in X$ to a binary class $c(x) \in \{-1, 1\}$
- A **linear classifier** uses:
  - **feature functions** $f(x) = (f_1(x), \ldots, f_m(x))$ and
  - **feature weights** $w = (w_1, \ldots, w_m)$
  to assign $x \in X$ to class $c(x) = \text{sign}(w \cdot f(x))$
  - $\text{sign}(y) = +1$ if $y > 0$ and $-1$ if $y < 0$
- Learn a linear classifier from **labeled training examples** $D = ((x_1, y_1), \ldots, (x_n, y_n))$ where $x_i \in X$ and $y_i \in \{-1, +1\}$

<table>
<thead>
<tr>
<th>$f_1(x_i)$</th>
<th>$f_2(x_i)$</th>
<th>$y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$+1$</td>
<td>$+1$</td>
</tr>
<tr>
<td>$+1$</td>
<td>$-1$</td>
<td>$+1$</td>
</tr>
<tr>
<td>$+1$</td>
<td>$+1$</td>
<td>$-1$</td>
</tr>
</tbody>
</table>
Nonlinear classifiers from linear learners

- Linear classifiers are straightforward but not expressive
- Idea: apply a nonlinear transform to original features

\[ h(x) = (g_1(f(x)), g_2(f(x)), \ldots, g_n(f(x))) \]

and learn a linear classifier based on \( h(x_i) \)

- A linear decision boundary in \( h(x) \) may correspond to a non-linear boundary in \( f(x) \)

- Example: \( h_1(x) = f_1(x), h_2(x) = f_2(x), h_3(x) = f_1(x)f_2(x) \)

<table>
<thead>
<tr>
<th>( f_1(x_i) )</th>
<th>( f_2(x_i) )</th>
<th>( f_1(x_i)f_2(x_i) )</th>
<th>( y_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Outline

Introduction
Linear and nonlinear classifiers
Kernels and classifiers
The kernelized perceptron learner
Conclusions
Linear classifiers using kernels

- Linear classifier decision rule: Given feature functions \( f \) and weights \( w \), assign \( x \in \mathcal{X} \) to class
  \[
c(x) = \text{sign}(w \cdot f(x))
  \]

- Linear kernel using features \( f \): for all \( u, v \in \mathcal{X} \)
  \[
  K(u, v) = f(u) \cdot f(v)
  \]

- The kernel trick: Assume \( w = \sum_{k=1}^{n} s_k f(x_k) \), i.e., the feature weights \( w \) are represented implicitly by examples \((x_1, \ldots, x_n)\). Then:
  \[
c(x) = \text{sign}\left(\sum_{k=1}^{n} s_k f(x_k) \cdot f(x)\right)
  = \text{sign}\left(\sum_{k=1}^{n} s_k K(x_k, x)\right)
  \]
Kernels can implicitly transform features

- **Linear kernel:** For all objects $u, v \in \mathcal{X}$

  $$K(u,v) = \mathbf{f}(u) \cdot \mathbf{f}(v) = f_1(u)f_1(v) + f_2(u)f_2(v)$$

- **Polynomial kernel:** (of degree 2)

  $$K(u,v) = (\mathbf{f}(u) \cdot \mathbf{f}(v))^2$$
  $$= f_1(u)^2f_1(v)^2 + 2f_1(u)f_1(v)f_2(u)f_2(v) + f_2(u)^2f_2(v)^2$$
  $$= (f_1(u)^2, \sqrt{2}f_1(u)f_2(u), f_2(u)^2)$$
  $$\cdot (f_1(v)^2, \sqrt{2}f_1(v)f_2(v), f_2(v)^2)$$

- So a degree 2 polynomial kernel is equivalent to a linear kernel with transformed features:

  $$\mathbf{h}(x) = (f_1(x)^2, \sqrt{2}f_1(x)f_2(x), f_2(x)^2)$$
Kernelized classifier using polynomial kernel

- **Polynomial kernel**: (of degree 2)

\[
K(u, v) = (f(u) \cdot f(v))^2
\]

\[= h(u) \cdot h(v), \text{ where:}
\]

\[
h(x) = (f_1(x)^2, \sqrt{2}f_1(x)f_2(x), f_2(x)^2)
\]

<table>
<thead>
<tr>
<th>(f_1(x_i))</th>
<th>(f_2(x_i))</th>
<th>(y_i)</th>
<th>(h_1(x_i))</th>
<th>(h_2(x_i))</th>
<th>(h_3(x_i))</th>
<th>(s_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>(\sqrt{2})</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>-(\sqrt{2})</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>-(\sqrt{2})</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>(\sqrt{2})</td>
<td>+1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Feature weights: 0 \(-2\sqrt{2}\) 0
Gaussian kernels and other kernels

- A “Gaussian kernel” is based on the distance $||f(u) - f(v)||$ between feature vectors $f(u)$ and $f(v)$

$$K(u, v) = \exp(-||f(u) - f(v)||^2)$$

- This is equivalent to a linear kernel in an infinite-dimensional feature space, but still easy to compute

⇒ *Kernels make it possible to easily compute over enormous (even infinite) feature spaces*

- There’s a little industry designing specialized kernels for specialized kinds of objects
Mercer’s theorem

- Mercer’s theorem: every continuous symmetric positive semi-definite kernel is a linear kernel in some feature space
  - this feature space may be infinite-dimensional
- This means that:
  - feature-based linear classifiers can often be expressed as kernel-based classifiers
  - kernel-based classifiers can often be expressed as feature-based linear classifiers
Outline

Introduction

Linear and nonlinear classifiers

Kernels and classifiers

The kernelized perceptron learner

Conclusions
The perceptron learner

- The perceptron is an error-driven learning algorithm for learning linear classifier weights $w$ for features $f$ from data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Algorithm:
  
  \[
  \begin{align*}
  &\text{set } w = 0 \\
  &\text{for each training example } (x_i, y_i) \in \mathcal{D} \text{ in turn:} \\
  &\quad \text{if } \text{sign}(w \cdot f(x_i)) \neq y_i: \\
  &\quad\quad \text{set } w = w + y_i f(x_i)
  \end{align*}
  \]
- The perceptron algorithm always chooses weights that are a linear combination of $\mathcal{D}$’s feature vectors

\[
  w = \sum_{k=1}^{n} s_k f(x_k)
\]

If the learner got example $(x_k, y_k)$ wrong then $s_k = y_k$, otherwise $s_k = 0$
Kernelizing the perceptron learner

- Represent \( w \) as linear combination of \( \mathcal{D}'s \) feature vectors

\[
    w = \sum_{k=1}^{n} s_k f(x_k)
\]

i.e., \( s_k \) is weight of training example \( f(x_k) \)

- Key step of perceptron algorithm:
  
  \[
  \text{if } \text{sign}(w \cdot f(x_i)) \neq y_i: \quad \text{set } w = w + y_i f(x_i) \]

becomes:

\[
\text{if } \text{sign}(\sum_{k=1}^{n} s_k f(x_k) \cdot f(x_i)) \neq y_i: \quad \text{set } s_i = s_i + y_i
\]

- If \( K(x_k, x_i) = f(x_k) \cdot f(x_i) \) is linear kernel, this becomes:

\[
\text{if } \text{sign}(\sum_{k=1}^{n} s_k K(x_k, x_i)) \neq y_i: \quad \text{set } s_i = s_i + y_i
\]
Kernelized perceptron learner

- The kernelized perceptron maintains weights \( s = (s_1, \ldots, s_n) \) of training examples \( D = ((x_1, y_1), \ldots, (x_n, y_n)) \)
  - \( s_i \) is the weight of training example \((x_i, y_i)\)
- Algorithm:
  
  \[
  \text{set } s = 0 \\
  \text{for each training example } (x_i, y_i) \in D \text{ in turn:} \\
  \quad \text{if } \text{sign}(\sum_{k=1}^{n} s_k K(x_k, x_i)) \neq y_i:
  \quad \quad \text{set } s_i = s_i + y_i
  \]
- If we use a *linear kernel* then kernelized perceptron *makes exactly the same predictions* as ordinary perceptron
- If we use a *nonlinear kernel* then kernelized perceptron makes exactly the same predictions as ordinary perceptron *using transformed feature space*
Gaussian-regularized MaxEnt models

- Given data $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$, the weights $w$ that maximize the Gaussian-regularized conditional log likelihood are:

$$\hat{w} = \arg\min_w Q(w) \text{ where:}$$

$$Q(w) = -\log L_\mathcal{D}(w) + \alpha \sum_{k=1}^m w_k^2$$

$$\frac{\partial Q}{\partial w_j} = \sum_{i=1}^n -\left(f_j(x_i, y_i) - E_w[f_j | x_i]\right) + 2\alpha w_j$$

- Because $\frac{\partial Q}{\partial w_j} = 0$ at $w = \hat{w}$, we have:

$$\hat{w}_j = \frac{1}{2\alpha} \sum_{i=1}^n \left(f_j(y_i, x_i) - E_{\hat{w}}[f_j | x_i]\right)$$
Gaussian-regularized MaxEnt can be kernelized

\[ \hat{w}_j = \frac{1}{2\alpha} \sum_{i=1}^{n} (f_j(y_i, x_i) - E_{\hat{w}}[f_j | x_i]) \]

\[ E_{\hat{w}}[f | x] = \sum_{y \in Y} f(y, x) P_{\hat{w}}(y | x), \text{ so:} \]

\[ \hat{w} = \sum_{x \in X_D} \sum_{y \in Y} \hat{s}_{y,x} f(y, x) \text{ where:} \]

\[ \hat{s}_{y,x} = \frac{1}{2\alpha} \sum_{i=1}^{n} \mathbb{I}(x, x_i)(\mathbb{I}(y, y_i) - P_{\hat{w}}(y, x)) \]

\[ X_D = \{ x_i | (x_i, y_i) \in D \} \]

⇒ the optimal weights \( \hat{w} \) are a linear combination of the feature values of \((y, x)\) items for \(x\) that appear in \(D\)
Outline

Introduction

Linear and nonlinear classifiers

Kernels and classifiers

The kernelized perceptron learner

Conclusions
Conclusions

- Many algorithms have *dual forms* using feature and kernel representations
- For any feature representation there is an equivalent kernel
- For any sensible kernel there is an equivalent feature representation
  - but the feature space may be infinite dimensional
- There can be substantial computational advantages to using features or kernels
  - many training examples, few features
    ⇒ features may be more efficient
  - many features, few training examples
    ⇒ kernels may be more efficient
- *Kernels make it possible to compute with very large (even infinite-dimensional) feature spaces, but each classification requires comparing to a potentially large number of training examples*