A brief introduction to kernel classifiers

Mark Johnson
Brown University

October 2009

Outline

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

$\Rightarrow$ 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
    $\Rightarrow$ feature version may be more efficient
  - few training examples, many features
    $\Rightarrow$ kernel version may be more efficient

Outline

Linear classifiers

- A *classifier* is a function c that maps an example $x \in \mathcal{X}$ to a binary class $c(x) \in \{-1, 1\}$
- A *linear classifier* uses:
    - *feature functions* $\mathbf{f}(x) = (f_1(x), \ldots, f_m(x))$ and
    - *feature weights* $\mathbf{w} = (w_1, \ldots, w_m)$

  to assign $x \in \mathcal{X}$ to class $c(x) = \text{sign}(\mathbf{w} \cdot \mathbf{f}(x))$
    - $\text{sign}(y) = +1$ if $y > 0$ and -1 if $y < 0$
- Learn a linear classifier from *labeled training examples* $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$ where $x_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$

$f_1(x_i)$	$f_2(x_i)$	y_i
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	-1

Nonlinear classifiers from linear learners

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

$$\mathbf{h}(x) \quad = \quad (g_1(\mathbf{f}(x)), g_2(\mathbf{f}(x)), \ldots, g_n(\mathbf{f}(x)))$$

  and learn a linear classifier based on $\mathbf{h}(x_i)$
- A linear decision boundary in $\mathbf{h}(x)$ may correspond to a *non-linear boundary* in $\mathbf{f}(x)$
- Example: $h_1(x) = f_1(x), h_2(x) = f_2(x), h_3(x) = f_1(x)f_2(x)$

$f_1(x_i)$	$f_2(x_i)$	$f_1(x_i)f_2(x_i)$	y_i
-1	-1	$+1$	-1
-1	$+1$	-1	$+1$
$+1$	-1	-1	$+1$
$+1$	$+1$	$+1$	-1

Outline

Linear classifiers using kernels

- Linear classifier decision rule: Given feature functions $\mathbf{f}$ and weights $\mathbf{w}$, assign $x \in \mathcal{X}$ to class

$$c(x) \;=\; \mathrm{sign}(\mathbf{w} \cdot \mathbf{f}(x))$$

- Linear kernel using features $\mathbf{f}$: for all $u, v \in \mathcal{X}$

$$K(u, v) \;=\; \mathbf{f}(u) \cdot \mathbf{f}(v)$$

- The *kernel trick*: Assume $\mathbf{w} = \sum_{k=1}^{n} s_k \, \mathbf{f}(x_k)$, i.e., the feature weights $\mathbf{w}$ are *represented implicitly* by examples $(x_1, \ldots, x_n)$. Then:

$$\begin{aligned} c(x) &= \mathrm{sign}(\sum_{k=1}^{n} s_k \, \mathbf{f}(x_k) \cdot \mathbf{f}(x)) \\ &= \mathrm{sign}(\sum_{k=1}^{n} s_k \, K(x_k, x)) \end{aligned}$$

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)

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

- 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)

$$\begin{aligned} K(u,v) &= (\mathbf{f}(u)\cdot\mathbf{f}(v))^2 \\ &= \mathbf{h}(u)\cdot\mathbf{h}(v), \text{ where:} \\ \mathbf{h}(x) &= (f_1(x)^2, \sqrt{2}f_1(x)f_2(x), f_2(x)^2) \end{aligned}$$

$f_1(x_i)$	$f_2(x_i)$	y_i	$h_1(x_i)$	$h_2(x_i)$	$h_3(x_i)$	s_i
-1	-1	-1	$+1$	$\sqrt{2}$	$+1$	-1
-1	$+1$	$+1$	$+1$	$-\sqrt{2}$	$+1$	$+1$
$+1$	-1	$+1$	$+1$	$-\sqrt{2}$	$+1$	$+1$
$+1$	$+1$	-1	$+1$	$\sqrt{2}$	$+1$	-1
Feature weights			0	$-2\sqrt{2}$	0	

Gaussian kernels and other kernels

- A "Gaussian kernel" is based on the distance $||\mathbf{f}(u) - \mathbf{f}(v)||$ between feature vectors $\mathbf{f}(u)$ and $\mathbf{f}(v)$

$$K(u,v) \quad = \quad \exp(-||\mathbf{f}(u) - \mathbf{f}(v)||^2)$$

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

$\Rightarrow$ *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

The perceptron learner

- The perceptron is an error-driven learning algorithm for learning linear classifer weights $\mathbf{w}$ for features $\mathbf{f}$ from data $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$
- Algorithm:

```
set $\mathbf{w} = \mathbf{0}$
for each training example $(x_i, y_i) \in D$ in turn:
    if $\mathrm{sign}(\mathbf{w} \cdot \mathbf{f}(x_i)) \neq y_i$:
        set $\mathbf{w} = \mathbf{w} + y_i\,\mathbf{f}(x_i)$
```

- The perceptron algorithm always choses weights that are a linear combination of $\mathcal{D}$'s feature vectors

$$\mathbf{w} \quad = \quad \sum_{k=1}^{n} s_k\,\mathbf{f}(x_k)$$

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

Kernelizing the perceptron learner

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

$$\mathbf{w} = \sum_{k=1}^{n} s_k \, \mathbf{f}(x_k)$$

  i.e., s_k is weight of training example $\mathbf{f}(x_k)$
- Key step of perceptron algorithm:

```
if sign($\mathbf{w} \cdot \mathbf{f}(x_i)$) $\neq y_i$:
    set $\mathbf{w} = \mathbf{w} + y_i \, \mathbf{f}(x_i)$
```

  becomes:

```
if sign($\sum_{k=1}^{n} s_k \, \mathbf{f}(x_k) \cdot \mathbf{f}(x_i)$) $\neq y_i$:
    set $s_i = s_i + y_i$
```

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

```
if sign($\sum_{k=1}^{n} s_k \, K(x_k, x_i)$) $\neq y_i$:
    set $s_i = s_i + y_i$
```

Kernelized perceptron learner

- The kernelized perceptron maintains weights $\mathbf{s} = (s_1, \ldots, s_n)$ of training examples $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$
  - s_i is the weight of training example (x_i, y_i)
- Algorithm:

```
set $\mathbf{s} = \mathbf{0}$
for each training example $(x_i, y_i) \in D$ in turn:
    if sign($\sum_{k=1}^{n} s_k \, K(x_k, x_i)$) $\neq y_i$:
        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 $\mathbf{w}$ that maximize the *Gaussian-regularized conditional log likelihood* are:

$$
\begin{aligned}
\widehat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} \, Q(\mathbf{w}) \text{ where:} \\
Q(\mathbf{w}) &= -\log L_{\mathcal{D}}(\mathbf{w}) + \alpha \sum_{k=1}^{m} w_k^2 \\
\frac{\partial Q}{\partial w_j} &= \sum_{i=1}^{n} -(f_j(x_i, y_i) - \mathrm{E}_{\mathbf{w}}[f_j \mid x_i]) + 2\alpha w_j
\end{aligned}
$$

- Because $\partial Q / \partial w_j = 0$ at $\mathbf{w} = \widehat{\mathbf{w}}$, we have:

$$
\widehat{w}_j = \frac{1}{2\alpha} \sum_{i=1}^{n} (f_j(y_i, x_i) - \mathrm{E}_{\widehat{\mathbf{w}}}[f_j \mid x_i])
$$

Gaussian-regularized MaxEnt can be kernelized

$$\begin{aligned}
\widehat{w}_j &= \frac{1}{2\alpha} \sum_{i=1}^{n} (f_j(y_i, x_i) - \mathrm{E}_{\widehat{\mathbf{w}}}[f_j \mid x_i]) \\
\mathrm{E}_{\mathbf{w}}[f \mid x] &= \sum_{y \in \mathcal{Y}} f(y, x)\, \mathrm{P}_{\mathbf{w}}(y \mid x), \text{ so:} \\
\widehat{\mathbf{w}} &= \sum_{x \in \mathcal{X}_{\mathcal{D}}} \sum_{y \in \mathcal{Y}} \hat{s}_{y,x} \mathbf{f}(y, x) \text{ where:} \\
\hat{s}_{y,x} &= \frac{1}{2\alpha} \sum_{i=1}^{n} \mathbb{I}(x, x_i)(\mathbb{I}(y, y_i) - \mathrm{P}_{\widehat{\mathbf{w}}}(y, x)) \\
\mathcal{X}_{\mathcal{D}} &= \{x_i \mid (x_i, y_i) \in \mathcal{D}\}
\end{aligned}$$

$\Rightarrow$ the optimal weights $\widehat{\mathbf{w}}$ are a linear combination of the feature values of (y, x) items for x that appear in D

Outline

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
    $\Rightarrow$ features may be more efficient
  - many features, few training examples
    $\Rightarrow$ 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*