Machine Learning (CSE 446): Kernel Methods

Noah Smith

© 2017

University of Washington
nasmith@cs.washington.edu

November 15, 2017
Can We Have Nonlinearity *and* Convexity?

<table>
<thead>
<tr>
<th></th>
<th>expressiveness</th>
<th>convexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear classifiers</td>
<td>😞</td>
<td>😞</td>
</tr>
<tr>
<td>Neural networks</td>
<td>👍</td>
<td>😞</td>
</tr>
</tbody>
</table>
Can We Have Nonlinearity *and* Convexity?

<table>
<thead>
<tr>
<th></th>
<th>expressiveness</th>
<th>convexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear classifiers</td>
<td>😞</td>
<td>😞</td>
</tr>
<tr>
<td>Neural networks</td>
<td>😊</td>
<td>😞</td>
</tr>
</tbody>
</table>

**Kernel** methods: a family of approaches that give us nonlinear decision boundaries without giving up convexity.
Notation

Let $\mathbf{x} = \langle x_1, x_2, \ldots, x_d \rangle$. 
Consider two binary features, $\phi_j$ and $\phi_{j'}$. A new conjunction feature can be defined by:

$$\phi_{j \land j'}(x) = \phi_j(x) \land \phi_{j'}(x) \quad \text{equivalently} \quad x_{d+1} = x_j \land x_{j'}$$
Consider two binary features, $\phi_j$ and $\phi_{j'}$. A new conjunction feature can be defined by:

$$\phi_{j \land j'}(x) = \phi_j(x) \land \phi_{j'}(x)$$

equivalently

$$x_{d+1} = x_j \land x_{j'}$$

Generalization: take the product of two features.
Conjunctive/Product Features

See slides 23–32 in the 10/13 “practical issues” lecture.

Consider two binary features, $\phi_j$ and $\phi_{j'}$. A new conjunction feature can be defined by:

$$\phi_{j \land j'}(x) = \phi_j(x) \land \phi_{j'}(x) \quad \text{equivalently} \quad x_{d+1} = x_j \land x_{j'}$$

Generalization: take the product of two features.
Bigger generalization: take all the products!

$$\phi(x) = \text{vector}(\langle 1; x \rangle \langle 1; x \rangle^\top)$$

$$= \langle 1, x_1, x_2, \ldots, x_d, x_1^2, x_1 \cdot x_2, \ldots, x_1 \cdot x_d, x_2^2, x_2 \cdot x_1, \ldots, x_2 \cdot x_d, \vdots, \vdots, \vdots, \vdots, \vdots, \vdots, x_{d-1} \cdot x_1, x_{d-1} \cdot x_2, \ldots, x_{d-1} \cdot x_d, x_d^2, x_d \cdot x_1, x_d \cdot x_2, \ldots, x_d \cdot x_d \rangle$$
The Kernel Trick

Some learning algorithms, like the perceptron, can be rewritten so that the only thing you do with feature vectors is take *inner products between them.*
The Kernel Trick

Some learning algorithms, like the perceptron, can be rewritten so that the only thing you do with feature vectors is take *inner products between them.*

Note that: $\phi(x) \cdot \phi(v)$

$$\begin{align*}
= & \ 1 \ + \ x_1 v_1 \ + \ x_2 v_2 \ + \cdots + \ x_d v_d \\
+ & \ x_1 v_1 \ + \ x_1^2 v_1^2 \ + \ x_1 x_2 v_1 v_2 \ + \cdots + \ x_1 x_d v_1 v_d \\
+ & \ x_2 v_2 \ + \ x_2 x_1 v_2 v_1 \ + \ x_2^2 v_2^2 \ + \cdots + \ x_2 x_d v_2 v_d \\
+ & \cdots \ + \ x_d v_d \ + \ x_d x_1 v_d v_1 \ + \ x_d x_2 v_d v_2 \ + \cdots + \ x_d^2 v_d^2 \\
= & \ 1 + 2 \cdot \sum_{j=1}^{d} x_j v_j + \sum_{j=1}^{d} \sum_{k=1}^{d} x_j x_k v_j v_k \\
= & \ 1 + 2 \cdot x \cdot v + (x \cdot v)^2 \\
= & \ (1 + x \cdot v)^2
\end{align*}$$
A **kernel** function (implicitly) computes:

$$K(x, v) = \phi(x) \cdot \phi(v)$$

for some $\phi$. Typically it is *cheap* to compute $K(\cdot, \cdot)$, and we never explicitly represent $\phi(v)$ for any vector $v$. 
Kernels

A kernel function (implicitly) computes:

\[ K(x, v) = \phi(x) \cdot \phi(v) \]

for some \( \phi \). Typically it is cheap to compute \( K(\cdot, \cdot) \), and we never explicitly represent \( \phi(v) \) for any vector \( v \).

Some kernels:

- quadratic: \( K^{quad}(x, v) = (1 + x \cdot v)^2 \)
- cubic: \( K^{cubic}(x, v) = (1 + x \cdot v)^3 \)
- polynomial: \( K^{poly}_p(x, v) = (1 + x \cdot v)^p \)
- radial basis function: \( K^{rbf}_\gamma(x, v) = \exp \left( -\gamma \|x - v\|^2_2 \right) \)
- hyperbolic tangent: \( \tilde{K}^{tanh}(x, v) = \tanh(1 + x \cdot v) \) (not a kernel)
- all conjunctions: \( K^{all \ conj}(x, v) = \prod_{j=1}^{d} (1 + x_j v_j) \) (for binary features)
Perceptron Learning Algorithm

**Data:** $D = \langle (x_n, y_n) \rangle_{n=1}^N$, number of epochs $E$

**Result:** weights $w$ and bias $b$

Initialize: $w = 0$ and $b = 0$;

for $e \in \{1, \ldots, E\}$ do

for $n \in \{1, \ldots, N\}$, in random order do

# predict

$\hat{y} = \text{sign} (w \cdot x_n + b)$;

if $\hat{y} \neq y_n$ then

# update

$w \leftarrow w + y_n \cdot x_n$;

$b \leftarrow b + y_n$;

end

end

end

return $w, b$

**Algorithm 1:** PerceptronTrain
Perceptron Representer Theorem

At every stage of learning, there exist \( \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle \) such that

\[
w = \sum_{n=1}^{N} \alpha_n \cdot x_n = \alpha^\top X
\]

In other words, \( w \) is always in the span of the training data.
Perceptron Learning Algorithm (with $\phi$)

**Data**: $D = \langle (x_n, y_n) \rangle_{n=1}^{N}$, number of epochs $E$

**Result**: weights $w$ and bias $b$

initialize: $w = 0$ and $b = 0$;

for $e \in \{1, \ldots, E\}$ do

  for $n \in \{1, \ldots, N\}$, in random order do

    # predict
    $\hat{y} = \text{sign} (w \cdot \phi(x_n) + b)$;

    if $\hat{y} \neq y_n$ then

      # update
      $w \leftarrow w + y_n \cdot \phi(x_n)$;
      $b \leftarrow b + y_n$;

    end

  end

end

return $w, b$

**Algorithm 2**: PERCEPTRONTRAIN with $\phi$ (explicit)
\hat{y} = \text{sign}(w \cdot \phi(x_n) + b)

= \text{sign} \left( \sum_{i=1}^{N} \alpha_i \cdot \phi(x_i) \cdot \phi(x_n) + b \right)

= \text{sign} \left( \sum_{i=1}^{N} \alpha_i \cdot K(x_i, x_n) + b \right)
The Update

\[ \mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} + y_n \cdot \phi(x_n) \]

\[
\sum_{i=1}^{N} \alpha_i^{\text{new}} \cdot \phi(x_i) \leftarrow \sum_{i=1}^{N} \alpha_i^{\text{old}} \cdot \phi(x_i) + y_n \cdot \phi(x_n)
\]

\[
\sum_{i \neq n} \alpha_i^{\text{new}} \cdot \phi(x_i) + \alpha_n^{\text{new}} \cdot \phi(x_n) \leftarrow \sum_{i \neq n} \alpha_i^{\text{old}} \cdot \phi(x_i) + (\alpha_n^{\text{old}} + y_n) \cdot \phi(x_n)
\]

\[
\alpha_n^{\text{new}} \cdot \phi(x_n) \leftarrow (\alpha_n^{\text{old}} + y_n) \cdot \phi(x_n)
\]

\[
\alpha_n^{\text{new}} \leftarrow \alpha_n^{\text{old}} + y_n
\]
\( \phi(x_n) \) is Never Explicitly Computed!

\[
\text{predict: } \hat{y} = \text{sign} \left( \sum_{i=1}^{N} \alpha_i \cdot K(x_i, x_n) + b \right)
\]

\[
\text{update: } \alpha_n^{(\text{new})} \leftarrow \alpha_n^{(\text{old})} + y_n
\]

We only calculate inner products of such vectors.
Kernelized Perceptron Learning Algorithm

Data: $D = \langle (x_n, y_n) \rangle_{n=1}^{N}$, number of epochs $E$

Result: weights $\alpha$ and bias $b$

initialize: $\alpha = 0$ and $b = 0$;

for $e \in \{1, \ldots, E\}$ do
  for $n \in \{1, \ldots, N\}$, in random order do
    # predict
    $\hat{y} = \text{sign} \left( \sum_{i=1}^{N} \alpha_i \cdot K(x_i, x_n) + b \right)$;
    if $\hat{y} \neq y_n$ then
      # update
      $\alpha_n \leftarrow \alpha_n + y_n$;
      $b \leftarrow b + y_n$;
    end
  end
end

return $\alpha, b$

Algorithm 3: KernelizedPerceptronTrain