CS480 Introduction to Machine Learning
Kernel Methods

Edith Law
Overview

• Revisiting SVM
• What are Kernels and Why?
• Examples of Kernels
• What makes a valid kernel?
• Kernelizing Algorithms
• Applications of Kernels
Overview

• **Revisiting SVM**
  • What are Kernels and Why?
  • Examples of Kernels
  • What makes a valid kernel?
  • Kernelizing Algorithms
  • Applications of Kernels
Revisiting Support Vector Machines
Non-Linearity Separable Data

• A linear boundary might be too simple to capture the data.

• Option 1: Relax the constraints and allow some points to be misclassified by the margin.

• Option 2: Allow a nonlinear decision boundary in the input space by finding a linear decision boundary in an expanded space (similar to adding polynomial terms in linear regression).
  
  – Here \( x_i \) is replaced by \( \phi(x_i) \), where \( \phi \) is called a feature mapping.
Margin Optimization in Feature Space

Replacing $x_i$ by $\phi(x_i)$, the SVM optimization problem for $w$ becomes:

**Primal form**

$$\min_{w,b} \frac{1}{2} \lVert w \rVert^2$$

subject to

$$y_i(w^T \phi(x_i) + b) \geq 1, \quad i = 1, \ldots, n$$

**Dual form**

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\phi(x_i)^T \phi(x_j))$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, n$$

$$\sum_{i} \alpha_i y_i = 0, \quad i = 1, \ldots, n$$
Feature Space Solution

The optimal weights, in the expanded feature space, are

\[ w^* = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) \]

Classification of an input \( x \) is given by:

\[ h_{w,b}(\hat{x}) = \text{sign}\left( \sum_{i=1}^{n} \alpha_i y_i (\phi(x_i)^T \phi(\hat{x})) + b \right) \]

Note that to solve the SVM optimization problem in dual form and to make a prediction, we only ever need to compute dot-products of feature vectors.
Overview

• Revisiting SVM
• **What are Kernels and Why?**
  • Examples of Kernels
  • What makes a valid kernel?
• Kernelizing Algorithms
• Applications of Kernels
Feature Mapping -> Non-Linear Decision Boundary

Linear models are easy to understand and optimize, but they can only learn simple decision boundaries.

We saw in an earlier lecture that we can add higher order terms to a linear regression problem. For example, using $x, x^2, x^3$ as features we get a cubic function.

The feature mapping $\phi$ maps attributes (original features) to features.

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$
Data may be separable in higher dimensions

\[ x = \{ x_1, x_2 \} \]

\[ z = \{ x, x^2 \} \]
Feature Mapping can be Expensive

Depending on the transformation, the feature mapping $\phi$ can be very expensive to compute.

Imagine getting a linear model to behave non-linearly by transforming the input to add all feature combinations:

$$
\phi(x) = \langle 1, 2x_1, 2x_2, 2x_3, \ldots, 2x_m, \\
x_1^2, x_1x_2, x_1, x_3, \ldots, x_1x_m \\
x_2x_1, x_2^2, x_2x_3, \ldots, x_2x_m \\
x_3x_1, x_3x_2, x_3^2, \ldots, x_3x_m \\
\ldots, \\
x_mx_1, x_mx_2, x_mx_3, \ldots, x_m^2 \rangle
$$
Training Algorithms on Expanded Feature Space

Training a classifier on this expanded feature space is problematic.

1. **computationally expensive**: if the learning algorithm scales linearly in the number of features, then you have squared the computational power and memory you need.

2. **statistical**: you will need quadratically many training examples in order to avoid overfitting.
Kernel Functions

A **kernel** is any function \( K = \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R} \) which corresponds to a dot product for some feature mapping \( \phi \), where \( m \) is the dimension of the feature vector representing the example \( x_i \).

\[
K(x_1, x_2) = \phi(x_1) \cdot \phi(x_2) \text{ for some } \phi
\]

Conversely, by choosing feature mapping \( \phi \), we implicitly choose a kernel function.

Recall that \( \phi(x_1) \cdot \phi(x_2) = \cos \angle(x_1, x_2) \), where \( \angle \) denotes the angle between the vectors, so a kernel function can be thought of as a notion of **similarity**.

But why is this dot product of the feature mapping useful?
Kernels: Example

Consider a feature space for 2-dimensional polynomials of degree 2.

\[
\phi(x) = \left[ x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1 \right]
\]

The inner product between two points in this feature space is:

\[
K(x, z) = \phi(x)^T \phi(z) = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1
\]

\[
= (x_1z_1 + x_2z_2 + 1)^2
\]

**Complexity:** 3 multiplications instead of 6!
Kernels: Example

Consider a feature space for 2-dimensional polynomials of degree 3.

\[ \phi(x) = \left[ x_1^3, x_2^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, \sqrt{3}x_1^2, \sqrt{3}x_2^2, \sqrt{6}x_1x_2, \sqrt{3}x_1, \sqrt{3}x_2, 1 \right] \]

The inner product between two points in this feature space is:

\[
K(x, z) = \phi(x)^T \phi(z) = x_1^3z_1^3 + x_2^3z_2^3 + 3x_1^2x_2z_1^2z_2 + 3x_1x_2^2z_1z_2^2 + 3x_1^2z_1^2 + 3x_2^2z_2^2 \\
+ 6x_1x_2z_1z_2 + 3x_1z_1 + 3x_2z_2 + 1 \\
= (x_1z_1 + x_2z_2 + 1)^3
\]

**Complexity:** 3 multiplications instead of 10!
Kernels: Example (Quadratic Kernel)

The feature mapping vector \( \phi(x) \) include all squares of elements and all cross terms.

\[
\phi(x) = \begin{bmatrix}
x_1x_1 \\
x_1x_2 \\
\vdots \\
x_1x_m \\
x_2x_1 \\
x_2x_2 \\
\vdots \\
x_2x_m \\
x_mx_1 \\
x_mx_2 \\
\vdots \\
x_mx_m 
\end{bmatrix}
\]

\[
K(x, z) = \phi(x)^T \phi(z)
= \sum_{i,j=1}^{m} (x_i x_j)(z_i z_j)
= \sum_{i=1}^{m} \sum_{j=1}^{m} x_i x_j z_i z_j
= \left( \sum_{i=1}^{m} x_i z_i \right) \left( \sum_{j=1}^{m} x_j z_j \right)
= (x^T z)^2
\]

**Important:** the \( K(x, z) = (x^T z)^2 \) only takes \( O(m) \) whereas computing \( \phi \) takes \( O(m^2) \).
Kernels: Example (Quadratic Kernel)

Consider this: \( K(x, z) = (x^T z + c)^2 \)

\[
K(x, z) = \left( \sum_{i=1}^{m} x_i z_i + c \right)^2 = \left( \sum_{i=1}^{m} (x_i z_i + c) \right) \left( \sum_{i=1}^{m} (x_i z_i + c) \right)
\]

\[
= \sum_{i=1}^{m} \sum_{j=1}^{m} (x_i z_i + c)(x_j z_j + c)
\]

\[
= \sum_{i=1}^{m} \sum_{j=1}^{m} x_i z_i x_j z_j + c x_i z_i + c x_j z_j + c^2
\]

\[
= c^2 + \sum_{i=1}^{m} \sum_{j=1}^{m} (x_i x_j)(z_i z_j) + 2c \sum_{i=1}^{m} (\sqrt{2c} x_i)(\sqrt{2c} z_i)
\]

More generally, \( K(x, z) = (x^T z + c)^R \) implicit computes the dot product of examples in high dimensional space in \( O(m) \) time, whereas calculating \( \phi \) directly would take \( O(m^R) \) time.

\[
\phi(x) = \begin{bmatrix}
  x_1 x_1 \\
  x_1 x_2 \\
  \vdots \\
  x_1 x_m \\
  x_2 x_1 \\
  \vdots \\
  x_m x_m \\
  \sqrt{2c} x_1 \\
  \vdots \\
  \sqrt{2c} x_m \\
  c
\end{bmatrix}
\]
Polynomial Kernel: Computational Cost

More generally, $K(x, z) = (x^T z)^d$ is a kernel, for any positive integer $d$:

$$K(x, z) = \left( \sum_{i=1}^{m} x_i z_i \right)^d$$

If we expanded the sum above in the naïve way, we get $m^d$ terms.

Terms are monomials (products of $x_i$) with total power equal to $d$.

As the number of terms grows as $O(m^d)$, so the computational cost is exponentially worse.
The Kernel Trick

Instead of explicitly expanding the feature space, the **kernel trick** to

- rewrite algorithms (e.g., put SVM in dual form) so that they only depend on dot products between two examples.
- replace dot product $\phi(x)^T \phi(z)$ by a kernel function $K(x, z)$ which computes the dot product implicitly.

With this method, you can compute $\phi(x) \cdot \phi(z)$ in exactly the same amount of time you can compute $x \cdot z$, i.e., $O(m)$. 
SVM Primal Form with Feature Mapping

\[ K(x, z) = \left( \sum_{i=1}^{m} x_i z_i \right)^d \]

If we use the primal form of the SVM, each term gets a weight.

**Primal form**

\[
\begin{aligned}
\min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\
\text{s.t.} \quad & y_i(w^T \phi(x_i) + b) \geq 1, \quad i = 1, \ldots, n
\end{aligned}
\]

**Curse of dimensionality**: it is very expensive both to optimize and to predict with an SVM in primal form.
If we work with the dual, we do not have to ever compute the feature mapping $\phi$. We just compute the similarity kernel $K$.

$K$ can be evaluated $O(m)$ time = **Big Savings**!
Kernelizing Algorithms

This technique is powerful: we can essentially rewrite any algorithm so that they only depend on the kernel product between data points, and never on the actual data points themselves. It enables the development of a large number of non-linear algorithms for “free”.

Many other machine learning algorithms have a “dual formulation”, in which dot-products of features can be replaced by kernels.

Examples:
• Perceptron
• Logistic regression
• Linear regression
Overview

• Revisiting SVM
• What are Kernels and Why?
  • **Examples of Kernels**
• What makes a valid kernel?
• Kernelizing Algorithms
• Applications of Kernels
RBF/Gaussian Kernel

Suppose $x \in \mathbb{R}^2$

Given $x$, compute features depending on proximity to $z$.

- $x \approx z$: similarity is high
- if $x$ far away from $z$: similarity is low

$$K(x, z) = \exp \left( - \frac{||x - z||^2}{2\sigma^2} \right)$$

$K$ is close to 1 when $x$ and $z$ are close, and near 0 when $x$ and $z$ are far apart.
RBF/Gaussian Kernels

An example of kernels that depend primarily on the distance between two input points.

Has the property that they are big when two points are similar, and decrease monotonically as distance becomes larger.

Use SVM software packages to solve for parameters.
• choice of parameter C
• choice of kernel and kernel parameters
• e.g., choose $\sigma^2$ if we use the Gaussian Kernel

$$f = \exp\left( - \frac{||x - z||^2}{2\sigma^2} \right)$$

• need to perform feature scaling before using the Gaussian Kernel
RBF/Gaussian Kernels

\[ z = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \quad f = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right) \]

\( \sigma^2 = 1 \quad \sigma^2 = 0.5 \quad \sigma^2 = 3 \)
RBF/Gaussian Kernel

Note the non-linear decision boundary
RBF/Gaussian Kernel
SVM

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

linear kernel

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, \textbf{Gamma} = 0.01, \textbf{C}=1

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, **Gamma = 1, C=1**

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, **Gamma = 10, C=1**

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
RBF Kernel, $\textbf{Gamma} = 100$, $C=1$

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, Gamma = 0.01, C=1

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, Gamma = 0.01, $C=10$

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, Gamma = 0.01, \( C=1000 \)

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, Gamma = 0.01, $C=10000$

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
SVM

RBF Kernel, Gamma = 0.01, $C=100000$

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/
### Library of Kernels

More than 30 different kernel functions are widely used in machine learning algorithms: splines, Chi-square, histogram, wavelets, etc.

<table>
<thead>
<tr>
<th>Kernel Type</th>
<th>Kernel Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$K(x, z) = x \cdot z$</td>
<td></td>
</tr>
<tr>
<td>Polynomial</td>
<td>$K(x, z) = (1 + x \cdot z)^d$</td>
<td>feature expansion has all monomial terms of total power</td>
</tr>
<tr>
<td>Gaussian</td>
<td>$K(x, z) = \exp\left(-\frac{|x - z|^2}{2\sigma^2}\right)$</td>
<td>This kernel has an infinite-dimensional feature expansion, but dot-products can still be computed in $O(m)$ (where $m=$#features)</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>$K(x, z) = \tanh(c_1 x \cdot z + c_2)$</td>
<td>mimics the behaviour of a 2-layer neural network</td>
</tr>
</tbody>
</table>

---

http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/
Kernels Beyond SVM

• A lot of research related to defining kernel functions suitable to particular tasks / kinds of inputs (e.g. words, graphs, images).

• Many kernels are available:
  – Information diffusion kernels (Lafferty and Lebanon, 2002)
  – Diffusion kernels on graphs (Kondor and Jebara, 2003)
  – String kernels for text classification (Lodhi et al, 2002)
  – String kernels for protein classification (Leslie et al, 2002)
  – … and others!
Example: String Kernels

• Very important for DNA matching, text classification, ...

• Goal: classify strings (e.g., list of amino acids strung to form a protein)

• Solution: use a sliding window of length $k$ over the two strings that we want to compare.

• Within the fixed-size window we can do many things:
  – Count exact matches.
  – Weigh mismatches based on how bad they are.
  – Count certain markers, e.g. AGT.

• The kernel is the sum of these similarities over the two sequences.

http://sonnenburgs.de/soeren/publications/SonRaeRie07.pdf
Overview

• Revisiting SVM
• What are Kernels and Why?
• Examples of Kernels
  • What makes a valid kernel?
• Kernelizing Algorithms
• Applications of Kernels
What makes a valid kernel?

\[ K : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R} \]

Can we use any \( K \)? Or, do only certain \( K \) retain the properties that the algorithms are supposed to have, e.g., convergence, optimality etc?

**Answer #1**: \( K \) is a valid kernel if it corresponds to the inner product between two vectors.

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

Is \( K(x, z) = (x^T z)^2 \) a kernel? Yes!
What makes a valid kernel?

**Answer #2.**

Consider some finite set of $n$ points $x_1, \ldots, x_n$

Let a square $n \times n$ Kernel matrix $K$ be defined so that its $(i,j)$ entry is given by $K_{ij} = K(x_i, x_j)$

**Mercer Theorem.** Let $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. For $K$ to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x_1, \ldots, x_n\}, (n < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.
What makes a valid kernel?

**Symmetry:**

\[
K_{ij} = K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \phi(x_j)^T \phi(x_i) = K(x_j, x_i) = K_{j,i}
\]

**Positive Semi-Definite:**

This means that for all functions \( f \) that are square integrable other than the zero function, the following property holds:

\[
\int \int f(x)K(x, z)f(z)dx dz > 0 \quad \forall f \int f(x)^2 dx < \infty
\]

Why is this characterization useful? It is useful because it gives us an alternative way to construct kernel functions.
**Constructing New Kernels**

**Theorem 13** (Kernel Addition). If $K_1$ and $K_2$ are kernels, the $K$ defined by $K(x, z) = K_1(x, z) + K_2(x, z)$ is also a kernel.

**Proof of Theorem 13.** You need to verify the positive semi-definite property on $K$. You can do this as follows:

\[
\iint f(x)K(x, z)f(z)\,dx\,dz = \iint f(x)[K_1(x, z) + K_2(x, z)]\,f(z)\,dx\,dz
\]

(11.15) definition of $K$

\[
= \iint f(x)K_1(x, z)f(z)\,dx\,dz + \iint f(x)K_2(x, z)f(z)\,dx\,dz
\]

(11.16) distributive rule

\[
> 0 + 0
\]

(11.17) $K_1$ and $K_2$ are psd

from [D] Ch.11
Kernel Algebra

There are several ways to combine together kernel functions to make new valid (positive semi-definite and symmetric) kernel functions.

Assume that we have two valid kernels $K_1(x,z)$ and $K_2(x,z)$, we can generate a new kernel using these:

$$K(x, z) = K_1(x, z) + K_2(x, z)$$  \hspace{1cm} \text{addition}

$$K(x, z) = K_1(x, z)K_2(x, z)$$  \hspace{1cm} \text{multiplication}

$$K(x, z) = cK_1(x, z) \text{ for } c > 0$$  \hspace{1cm} \text{scaling}

$$K(x, z) = f(x)K_1(x, z)f(z) \text{ for any } f(x)$$  \hspace{1cm} \text{outer product}

These properties let us define a variety of new kind of kernels.
Techniques for Constructing New Kernels.

Given valid kernels $k_1(x, x')$ and $k_2(x, x')$, the following new kernels will also be valid:

$$k(x, x') = c k_1(x, x')$$  \hspace{1cm} (6.13)  \\
$$k(x, x') = f(x) k_1(x, x') f(x')$$ \hspace{1cm} (6.14)  \\
$$k(x, x') = q(k_1(x, x'))$$ \hspace{1cm} (6.15)  \\
$$k(x, x') = \exp(k_1(x, x'))$$ \hspace{1cm} (6.16)  \\
$$k(x, x') = k_1(x, x') + k_2(x, x')$$ \hspace{1cm} (6.17)  \\
$$k(x, x') = k_1(x, x') k_2(x, x')$$ \hspace{1cm} (6.18)  \\
$$k(x, x') = \phi(x), \phi(x')$$ \hspace{1cm} (6.19)  \\
$$k(x, x') = x^T A x'$$ \hspace{1cm} (6.20)  \\
$$k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b)$$ \hspace{1cm} (6.21)  \\
$$k(x, x') = k_a(x_a, x'_a) k_b(x_b, x'_b)$$ \hspace{1cm} (6.22)

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(x)$ is a function from $x$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $A$ is a symmetric positive semidefinite matrix, $x_a$ and $x_b$ are variables (not necessarily disjoint) with $x = (x_a, x_b)$, and $k_a$ and $k_b$ are valid kernel functions over their respective spaces.
Overview

• Revisiting SVM
• What are Kernels and Why?
• Examples of Kernels
• What makes a valid kernel?
• **Kernelizing Algorithms**
• Applications of Kernels
The Perceptron Algorithm (from Lecture 5)

Algorithm 5 \textbf{PerceptronTrain}(D, MaxIter)

\begin{algorithmic}
\State $w_d \leftarrow 0$, for all $d = 1 \ldots D$ \hfill \text{\textit{initialize weights}}
\State $b \leftarrow 0$ \hfill \text{\textit{initialize bias}}
\For{$iter = 1 \ldots \text{MaxIter}$}
\For{all $(x, y) \in D$}
\State $a \leftarrow \sum_{d=1}^{D} w_d x_d + b$ \hfill \text{\textit{compute activation for this example}}
\If{$ya \leq 0$}
\State $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ \hfill \text{\textit{update weights}}
\State $b \leftarrow b + y$ \hfill \text{\textit{update bias}}
\EndIf
\EndFor
\EndFor
\State \textbf{return} $w_0, w_1, \ldots, w_D, b$
\end{algorithmic}

Algorithm 6 \textbf{PerceptronTest}(w_0, w_1, \ldots, w_D, b, \hat{x})

\begin{algorithmic}
\State $a \leftarrow \sum_{d=1}^{D} w_d \hat{x}_d + b$ \hfill \text{\textit{compute activation for the test example}}
\State \textbf{return} $\text{SIGN}(a)$
\end{algorithmic}
The Perceptron Algorithm with Feature Mapping

Suppose now we want to introduce feature mapping to learn a non-linear decision boundary. Can we apply to the Kernel Trick?

No. We first have to rewrite the algorithm using dot products between examples, and remove the dependence on \( w \) and \( \phi \).
Kernelizing the Perceptron

**Perceptron Representer Theorem.** During a run of the perceptron algorithm, the weight vector $w$ can always be represented as a linear combination of the expanded training data $\phi(x_1), \ldots, \phi(x_N)$. That is, at any point in the algorithm, the weight vector can be written as:

$$w = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

**Proof** is provided in [D] Ch.11.
Kernelizing the Perceptron: Intuition

The update rule (line 6) for the weight vector is this:

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

We can show that we can represent \( w \) as a linear combination of \( \phi(x_i) \) at each step of the algorithm.

\[
    w = \sum_{i=1}^{n} \alpha_i \phi(x_i)
\]

Initially:
\[
\alpha_1, \alpha_2, \ldots, \alpha_n = 0
\]
\[
w = 0
\]

First update at \( x_j \):
\[
\alpha_j = \alpha_j + y_j
\]
\[
w \leftarrow \alpha_j \phi(x_j)
\]
\[
+ \sum_{i=1, i\neq j}^{n} \alpha_i \phi(x_i)
\]

Second update at \( x_k \):
\[
\alpha_k \leftarrow \alpha_k + y_k
\]
\[
w \leftarrow \alpha_j \phi(x_j) + \alpha_k \phi(x_k)
\]
\[
+ \sum_{i=1, i\neq j, k}^{n} \alpha_i \phi(x_i)
\]

...
Kernelizing the Perceptron

Now that we know that we can always write \( w = \sum_{i=1}^{n} \alpha_i \phi(x_i) \), we can rewrite the prediction rule on line 4 for the current example \( x \):

\[
a \leftarrow w \cdot \phi(x) + b
\]

such that it depends on only dot products between the data points, and never explicitly requires a weight vector, because:

\[
w \cdot \phi(x) + b = \left( \sum_{i=1}^{n} \alpha_i \phi(x_i) \right) \cdot \phi(x) + b = \sum_{i=1}^{n} \alpha_i [\phi(x_i) \cdot \phi(x)] + b
\]

nice!
Kernelized Perceptron

**Algorithm 30 KernelizedPerceptronTrain** \((D, \text{MaxIter})\)

1. \(\alpha \leftarrow 0, b \leftarrow 0\)  
   // initialize coefficients and bias
2. **for** iter = 1 \(\ldots\) MaxIter **do**
3.   **for all** \((x_n, y_n) \in D\) **do**
4.     \[a \leftarrow \sum_m \alpha_m \phi(x_m) \cdot \phi(x_n) + b\]  
      // compute activation for this example
5.     **if** \(y_na \leq 0\) **then**
6.       \[\alpha_n \leftarrow \alpha_n + y_n\]  
       // update coefficients
7.       \(b \leftarrow b + y\)  
       // update bias
8.     **end if**
9.   **end for**
10. **end for**
11. **return** \(\alpha, b\)
RBF Kernel

We can plug a RBF kernel function into a perceptron:

\[
f(\hat{x}) = \sum_{i=1}^{n} \alpha_i K(x_i, \hat{x}) + b
\]

Each training example gets to “vote” on the label of the test point \(x \).

The amount of “vote” of the \(n\)th example is proportional to the negative exponential of the distance between the test point and itself.
Overview

- Revisiting SVM
- What are Kernels and Why?
- Examples of Kernels
- What makes a valid kernel?
- Kernelizing Algorithms
- Applications of Kernels
Application of Kernels in NLP/Text Classification

Efficient Support Vector Classifiers for Named Entity Recognition

Hideki Isozaki and Hideto Kazawa
NTT Communication Science Laboratories
2-4 Hikari-dai, Seika-cho, Soraku-gun, Kyoto, 619-0237, Japan
{isozaki, kazawa}@culab.kecl.ntt.co.jp

Abstract
Named Entity (NE) recognition is a task in which proper nouns and numerical information are extracted from documents and are classified into categories such as person, organization, and date. It is a key technology of Information Extraction and Open-Domain Question Answering. First, we show that an NE recognizer based on Support Vector Machines (SVMs) gives better scores than conventional systems. However, off-the-shelf SVM classifiers are too inefficient for this task. Therefore, we present a method that makes the system substantially faster. This approach can also be applied to other similar tasks such as chunking and part-of-speech tagging. We also present an SVM-based feature selection method and an efficient training method.

1 Introduction
Named Entity (NE) recognition is a task in which proper nouns and numerical information in a document are detected and classified into categories such as person, organization, and date. It is a key technology of Information Extraction and Open-Domain Question Answering (Voorhees and Harman, 2000).

text classification using string kernels

Huma Lodhi  John Shawe-Taylor  Nello Cristianini
Department of Computer Science Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
{huma, john, nello, chrisw}@dcs.rhul.ac.uk

Abstract
We introduce a novel kernel for comparing two text documents. The kernel is an inner product in the feature space consisting of all subsequences of length k. A subsequence is any ordered sequence of k characters occurring in the text though not necessarily contiguous. The subsequences are weighted by an exponentially decaying factor of their full length in the text, hence emphasising those occurrences which are close to contiguous. A direct computation of this feature vector would involve a prohibitive amount of computation even for modest values of k, since the dimension of the feature space grows exponentially with k. The paper describes how despite this fact the inner product can be efficiently evaluated by a dynamic programming technique. A preliminary experimental comparison of the performance of the kernel compared with a standard word feature space kernel f61 is made showing encouraging...
What you should know

- The Kernel Trick and its benefits
- The connection between SVM dual formulation and kernels
- Examples of common kernels
- Kernelizing algorithms