Kernel Machines

Ying Wu

Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

http://www.eecs.northwestern.edu/~yingwu
From Linear to Nonlinear

- Linear classifiers are fundamental
- But they are limited
- In practice, we need to have nonlinear decision boundaries
- Can we generalize linear models to nonlinear?
Generalized Linear Discriminant Function

The linear discriminant can be generalized

\[ g(x) = \sum_{i=1}^{d'} a_i \phi_i(x) = a^T \Phi(x) = a^T y \]

where \( \phi_i(x) \) can be an arbitrary function of \( x \)

Example

\[ g(x) = a_1 + a_2 x + a_3 x^2 \]

\( \Phi(x) : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} \), maps the original feature space to another feature space

We wish the data becomes linearly separable in \( \mathbb{R}^{d'} \)

A hyperplane in \( \mathbb{R}^{d'} \) corresponds to a nonlinear surface in \( \mathbb{R}^d \)

How do we determine the form and the order of \( \Phi(x) \)?

How do we compute the discriminant?
Outline

The Kernel Trick

Kernel SVM

Kernel PCA

Kernel MDA
The Dimension of $\mathbb{R}^{d'}$

- What we wish is that the data become more linearly separable in $\mathbb{R}^{d'}$ after the mapping $\Phi(x) : \mathbb{R}^d \to \mathbb{R}^{d'}$
- Will $d' < d$ do the job?
- It seems to be very difficult, if not impossible, as we need to have a very discriminative $\Phi(x)$
- What if we have $d' > d$?
- The choices of $\Phi(x)$ can be quite flexible.
- No free lunch!
- It is possible that $d' \gg d$, even $d' = \infty$
- Now calculating $\Phi(x)$ is very difficult, if not impossible.
Kernel

- The kernel function is

\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]

- The kernel function \( K(x_i, x_j) \) of \( x_i \in \mathbb{R}^d \) and \( x_j \in \mathbb{R}^d \) is the inner product of the transformed features \( \Phi(x_i) \in \mathbb{R}^{d'} \) and \( \Phi(x_j) \in \mathbb{R}^{d'} \)

- This is significant!

- If we want to compute the inner product of two transformed features, we don’t have to do it in \( \mathbb{R}^{d'} \)

- Instead, we can simply do it in \( \mathbb{R}^d \)

- And more importantly, we don’t need to have to know the nonlinear mapping \( \Phi(x) \) explicitly!

- What a relief if \( d' = \infty \)!

- Defining the kernel function \( K(\cdot, \cdot) \) is enough

- \( K(\cdot, \cdot) \) defines \( \Phi(\cdot) \) implicitly
An Example: Polynomial Kernel

▶ If

\[ \Phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \]

▶ Then we have

\[
\Phi(x_a) \cdot \Phi(x_b) = \left[ x_{a1}^2 \sqrt{2}x_{a1}x_{a2} x_{a2}^2 \right] \left[ \begin{array}{c} x_{b1}^2 \\ \sqrt{2}x_{b1}x_{b2} \\ x_{b2}^2 \end{array} \right]
\]
\[
= x_{a1}^2 x_{b1}^2 + 2x_{a1}x_{a2}x_{b1}x_{b2} + x_{a2}^2 x_{b2}^2
\]
\[
= (x_a \cdot x_b)^2
\]
\[\triangleq K(x_a, x_b)\]
Can Any Function be Kernel?

- The inner product in $\mathbb{R}^{d'}$ is made possible (much simpler) in $\mathbb{R}^d$ by the Kernel function
- So, can any function be kernel?
- No, to be eligible for being a kernel function,
- Mercer's Theorem:

**Theorem**

$K(x, y)$ can be expanded into a uniformly convergent series:

$$K(x, y) = \sum_{i}^{\infty} \lambda_i \Phi_i(x) \Phi_i(y)$$

if and only if $\forall g(x) \in L^2$, i.e, $\int g^2(x) dx < \infty$,

$$\int K(x, y) g(x) g(y) \geq 0$$

is satisfied.
Commonly Used Kernel Functions

- There are some commonly used Kernels

\[ K(x, y) = (x \cdot y + 1)^n \quad \text{polynomial kernel} \]
\[ K(x, y) = \exp \left( -\frac{||x - y||^2}{2\sigma^2} \right) \quad \text{RBF kernel} \]
\[ K(x, y) = \tanh(\kappa x \cdot y - \delta) \]

- Note: RBF kernel maps the data to an infinite-dim space!
Outline

The Kernel Trick

Kernel SVM

Kernel PCA

Kernel MDA
Recall the Linear SVM

Let’s see the SVM discriminant function

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

Notice that \( \alpha_i = 0 \) for non-support vectors, so

\[ f(x) = w \cdot x + b = \sum_{x_i \in \{S.V.\}} \alpha_i y_i (x_i \cdot x) + b \]

It says the decision surface (i.e., the hyperplane) is fully determined by the “support vectors” that are located on the margin.

This is only related to the inner product of the input \( x \) and the support vectors \( x_i \in \{S.V.\} \).
Play the Kernel Trick!

- Using the idea of generalized linear discriminant $\mathbf{x} \rightarrow \Phi(\mathbf{x})$
- We have the nonlinear form

$$f(\mathbf{x}) = \sum_{\mathbf{x}_i \in \{S.V.\}} \alpha_i y_i (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) + b$$

- Note: we only have $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$
- Using the Kernel function

$$f(\mathbf{x}) = \sum_{\mathbf{x}_i \in \{S.V.\}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- Done!
- The trick: Replace $\mathbf{x}_i \cdot \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$
Outline

The Kernel Trick

Kernel SVM

Kernel PCA

Kernel MDA
Can We Do This?

PCA

KPCA
Recall PCA

Learning the principal components from \( \{x_1, \ldots, x_N\} \). W.l.g., we assume that data has been centered, i.e., \( \sum_{i=1}^{n} x_i = 0 \)

\[ \text{1. calculating } S = \sum_{k=1}^{N} x_k x_k^T = XX^T \]

\[ \text{2. eigenvalue decomposition } S = U^T \Sigma U \]

\[ \text{3. sorting } \lambda_i \text{ and } e_i \]

\[ \text{4. finding the bases } W = [e_1, e_2, \ldots, e_m] \]

Note: The components for \( x \) is

\[ y = W^T x, \text{ where } x \in \mathbb{R}^n \text{ and } y \in \mathbb{R}^m \]
A Closer Look at $\mathbf{w}$

- The eigenvector $\mathbf{w}$ is a linear combination of the data!

\[
\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i = \mathbf{Xa}, \quad \text{where} \quad \mathbf{a} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}
\]

- To see why, since $\lambda \mathbf{w} = \mathbf{Sw}$

\[
\begin{align*}
\mathbf{w} &= \frac{1}{\lambda} \mathbf{Sw} = \frac{1}{\lambda} \mathbf{XX}^T \mathbf{w} \\
&= \mathbf{X} \begin{bmatrix} \frac{1}{\lambda} \mathbf{X}^T \mathbf{w} \end{bmatrix} = \mathbf{Xa}
\end{align*}
\]
Let’s See How It is Important

- Let’s take \( <v, \lambda w> \)

\[
<v, \lambda w> = \lambda v^T w = \lambda v^T X a
\]

- Let’s take \( <v, Sw> \)

\[
<v, Sw> = v^T X X^T X a = (v^T X)(X^T X)a
\]

- What conclusion do you have?

\[
\lambda a = (X^T X)a
\]

- It says \( a \) is an eigenvector of \( X^T X \)

- This is the dual for \( XX^T \)!
Kernelization

- \( \forall x_i \), we map it to \( \Phi(x_i) \), and \( X \rightarrow Y \). So, we have

\[
S_{\phi} = YY^T
\]

- Note that, \( S_{\phi} \) can be a \( \infty \times \infty \) matrix!

- But \( Y^T Y \) is \( n \times n \), and it can be done easily. Why?

\[
K \triangleq Y^T Y = \begin{bmatrix}
\Phi^T(x_1) \\
\vdots \\
\Phi^T(x_n)
\end{bmatrix}
\begin{bmatrix}
\Phi(x_1) & \cdots & \Phi(x_n)
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\Phi^T(x_1)\Phi(x_1) & \cdots & \Phi^T(x_1)\Phi(x_n) \\
\cdots & \cdots & \cdots \\
\Phi^T(x_n)\Phi(x_1) & \cdots & \Phi^T(x_n)\Phi(x_n)
\end{bmatrix}
\]

\[
= \begin{bmatrix}
K(x_1, x_1) & \cdots & K(x_1, x_n) \\
\cdots & \cdots & \cdots \\
K(x_n, x_1) & \cdots & K(x_n, x_n)
\end{bmatrix}_{n \times n}
\]
Kernelization (cont.)

- Suppose \( w_\phi \) is the eigenvector of \( S_\phi \), we have

\[
w_\phi = \sum_{i=1}^{\phi} \alpha_i \Phi(x_i) = \Phi(X)a = Ya
\]

- Let’s play the kernel trick on

\[
\lambda a = (X^T X)a
\]

- What do we have?

\[
\lambda a = \Phi^T (X) \Phi(X)a = Ka
\]

- We do EVD on \( K \) to obtain \( a \)

- For a new input \( x \), its kernel principal components (KPC)

\[
z = w_\phi^T \Phi(x) = a^T Y^T \Phi(x) = a^T K(X, x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)
\]

- Isn’t it nice?
Summary Kernel PCA

Learning KPCA from \( \{x_1, \ldots, x_N\} \):

1. calculating \( m = \frac{1}{N} \sum_{k=1}^{N} x_k \)
2. centering \( D = [x_1 - m, \ldots, x_N - m] = [\bar{x}_1, \ldots, \bar{x}_n] \)
3. calculating \( K \), where \( K_{ij} = K(\bar{x}_i, \bar{x}_j) \)
4. eigenvalue decomposition

\[ K = U^T \Sigma U \]

5. sorting \( \lambda_i \) and \( a_i \)

Note: The \( i \)-th kernel principal components for \( x \) is

\[ z = a_i^T K(Y, x) = \sum_{j=1}^{n} \alpha_{ij} K(x_j, x) \]

Note: Fortunately, we only need to worry about the 1st KPC in most cases in practice.
Outline

The Kernel Trick

Kernel SVM

Kernel PCA

Kernel MDA
Multiple Discriminant Analysis

Now, we have $c$ number of classes:

- within-class scatter $S_w = \sum_{i=1}^{c} S_i$

- between-class scatter is a bit different from 2-class

$$S_b \triangleq \sum_{i=1}^{c} n_i (m_i - \mathbf{m})(m_i - \mathbf{m})^T$$

- total scatter

$$S_t \triangleq \sum_{\mathbf{x}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T = S_w + S_b$$

- MDA is to find a subspace with bases $\mathbf{W}$ that maximizes

$$J(\mathbf{W}) = \frac{|\tilde{S}_b|}{|\tilde{S}_w|} = \frac{|\mathbf{W}^T S_b \mathbf{W}|}{|\mathbf{W}^T S_w \mathbf{W}|}$$
In the New Feature Space

After the mapping $\Phi(\cdot) : \mathbf{X} \rightarrow \mathbf{Y}$, we need to

$$
\mathbf{V}^* = \text{arg max}_{\mathbf{V}} \frac{|\mathbf{V}^T \mathbf{S}_B^{\phi} \mathbf{V}|}{|\mathbf{V}^T \mathbf{S}_W^{\phi} \mathbf{V}|},
$$

where

$$
\mathbf{S}_B^{\phi} = \sum_{j=1}^{c} n_j (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^T,
$$

$$
\mathbf{S}_W^{\phi} = \sum_{j=1}^{c} \sum_{k=1}^{n_j} (\phi(x_k) - \mathbf{m}_j)(\phi(x_k) - \mathbf{m}_j)^T,
$$

with $\mathbf{m} = \frac{1}{n} \sum_{k=1}^{n} \Phi(x_k)$, $\mathbf{m}_j = \frac{1}{n_j} \sum_{k=1}^{n_j} \Phi(x_k)$, where $j = 1, \ldots, c$. 
As Before

- In the new space, the MDA basis
  \[ \mathbf{v} = \sum_{i=1}^{n} \alpha_i \Phi(x_i) = \mathbf{Ya} \]

- As before in KPCA, the projection of \( \Phi(x_k) \) on \( \mathbf{v} \) is
  \[ \mathbf{v}^T \Phi(x_k) = \mathbf{a}^T \mathbf{Y}^T \Phi(x_k) = \mathbf{a}^T \mathbf{K} (\mathbf{X}, x_k) \triangleq \mathbf{a}^T \xi_k \]

- For the center of each class
  \[
  \mathbf{v}^T \mathbf{m}_j = \mathbf{a}^T \frac{1}{n_j} \sum_{k=1}^{n_j} \begin{bmatrix}
  \phi^T(x_1) \phi(x_k) \\
  \vdots \\
  \phi^T(x_n) \phi(x_k)
  \end{bmatrix} \\
  = \mathbf{a}^T \begin{bmatrix}
  \frac{1}{n_j} \sum_{k=1}^{n_j} \mathbf{K}(x_1, x_k) \\
  \vdots \\
  \frac{1}{n_j} \sum_{k=1}^{n_j} \mathbf{K}(x_n, x_k)
  \end{bmatrix} \triangleq \mathbf{a}^T \mu_j
  \]
Kernel MDA

Let’s define

\[ K_B = \sum_{j=1}^{c} n_j (\mu_j - \mu)(\mu_j - \mu)^T \]

and

\[ K_W = \sum_{j=1}^{c} \sum_{k=1}^{n_j} (\xi_k - \mu_j)(\xi_k - \mu_j)^T \]

It is clear (you do it)

\[ v^T S_B^\phi v = a^T K_B a \]

and

\[ v^T S_W^\phi v = a^T K_W a \]

We end up with Kernel MDA

\[ A^* = \arg \max_A \frac{|A^T K_B A|}{|A^T K_W A|}, \]