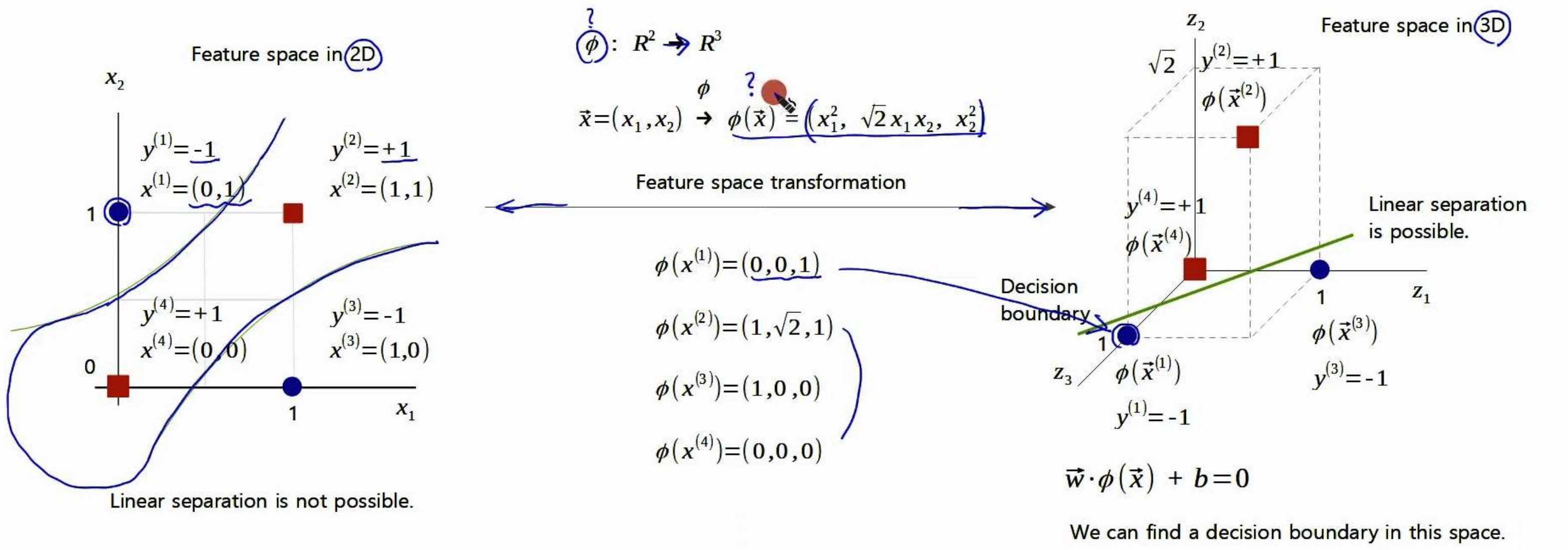
Nonlinear SVM Revisited

Thursday 23 May 2024 11:20 PM

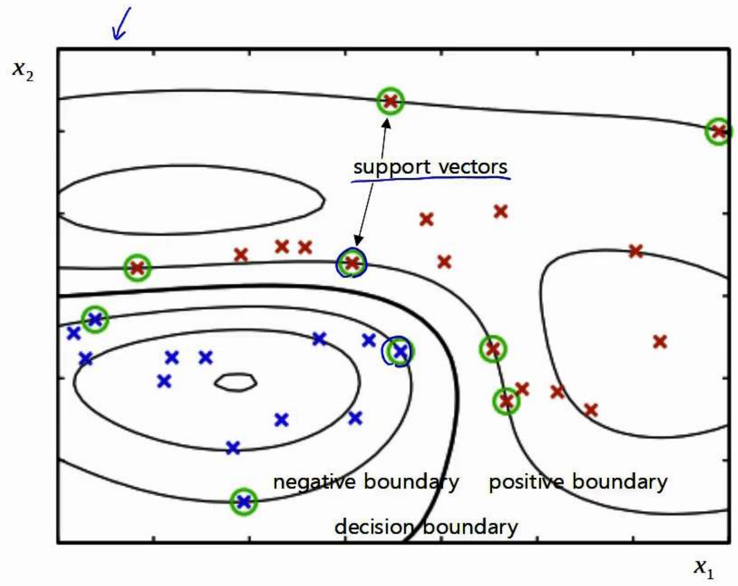
- If linear separation is not possible, a non-linear boundary can be obtained by converting the data into a space where linear separation is possible, then linearly separating the data and then converting it back to the original space. This is the idea of a non-linear SVM and does not actually convert the data into that space. Instead, we use a kernel trick to achieve this effect. The example below assumes that we know the function ϕ that converts the data from a 2D space into a 3D space. In reality, we neither know nor need to know this function ϕ.



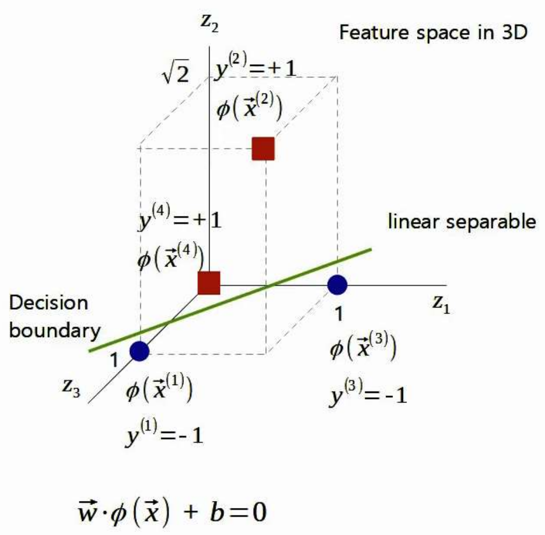
- Once a linear decision boundary is determined in the transformed space, a nonlinear decision boundary is created in the original space.

Source: Christopher M. Bishop, 2006, Pattern Recognition and Machine Learning. p.331.

Figure 7.2
Example of synthetic data from two classes in two dimensions showing contours of constant y(x) obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.



- The decision boundary can be created by applying the existing soft margin SVM to a linearly separable space.
- Just replace x with ϕ(x) in the formulas for the existing soft margin SVM.
- We can find the solution to the Lagrange dual function using the transformed data ϕ(x) instead of the data x in the original space.
- It is not possible to find ϕ(x) and w. However, the value of $\phi(x_i)\cdot\phi(x_j)$ and b can be found. Even if we don't know ϕ(x), we can determine the decision boundary by just knowing $\phi(x_i)\cdot\phi(x_j)$ and b. This method is called a kernel trick.



- Primal Lagrange

These values are unknown.

?

$$L_p = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\lambda_i\{y^{(i)}(\vec{w}\cdot\phi(\vec{x}^{(i)})+b)-1+\xi\} - \sum_{i=1}^{N}\mu_i\xi_i$$

$$\frac{\partial L_p}{\partial \vec{w}} = 0 \to \vec{w} = \sum_{i=1}^{N}\lambda_i y^{(i)}\phi(\vec{x}^{(i)}) \qquad \frac{\partial L_p}{\partial b} = 0 \to \sum_{i=1}^{N}\lambda_i y^{(i)} = 0$$

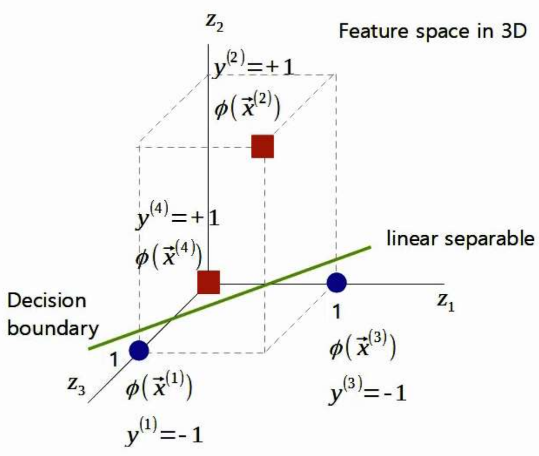
- Dual Lagrange

$$L_D = \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y^{(i)}y^{(j)}\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})$$

Since we can find this value using a kernel function, we can solve QP to find λ and find b as before. Since we do not know ϕ(x), we cannot find w.

$$\text{constraints:}\quad 0\le\lambda_i\le C,\quad \sum_{i=1}^{N}\lambda_i y^{(i)} = 0$$

Decision function



$$\vec{w}\cdot\phi(\vec{x}) + b = 0$$

- Since we do not know $\phi(x)$, we cannot find w. However, if we know $\phi(x_i)\phi(x)$, we can find $w\phi(x)$.

$$\vec{w}\cdot\phi(\vec{x})=\sum_{s\in SV}\lambda_s y^{(s)}\phi(\vec{x}^{(s)})\cdot\phi(\vec{x}) \leftarrow \vec{w}=\sum_{s\in SV}\lambda_s y^{(s)}\phi(\vec{x}^{(s)}) \leftarrow \vec{w}=\sum_{i=1}^{N}\lambda_i y^{(i)}\phi(\vec{x}^{(i)})$$

$$\vec{w} = \sum_{i=1}^{N}\lambda_i y^{(i)}\cdot\vec{x}^{(i)}$$

- Find b using the support vectors (SV).

$$y^{(i)}\Big(\sum_{s\in SV}\lambda_s y^{(s)}\phi(\vec{x}^{(s)})\cdot\phi(\vec{x}^{(i)})+b\Big)=1 \leftarrow y^{(i)}(\vec{w}\,\phi(\vec{x}^{(i)})+b)-1=0$$

$$y^{(i)}(\vec{w}\cdot x^{(i)}+b)-1=0$$

where SV denotes the set of indices of the support vectors. Although we can solve this equation for b using an arbitrarily chosen support vector $x^{(i)}$, a numerically more stable solution is obtained by first multiplying through by $y^{(i)}$, making use of $y^{(i)2} = 1$, and then averaging these equations over all support vectors and solving for b to give (source: Bishop, Pattern Recognition and Machine Learning, p.330, equation 7.18)

$$b=\frac{1}{N_{SV}}\sum_{i\in SV}\Big(y^{(i)}-\sum_{s\in SV}\lambda_s y^{(s)}\phi(\vec{x}^{(s)})\cdot\phi(\vec{x}^{(i)})\Big)$$

$\vec{w}\cdot\vec{x}$

for linear SVM:

$$b^{*} = mean(y^{(i)}-\vec{w}\cdot\vec{x}^{(i)})$$

(i: support vectors)

- Decision function

(test data)

$$\hat{y}= sign\Big[\sum_{i\in SV}\lambda_i y^{(i)}\phi(\vec{x}^{(i)})\cdot\phi(\vec{x})+b\Big]$$

$\vec{w}\cdot\phi(\vec{x})$

$\hat{y}>0 \rightarrow \hat{y}=+1$

$\hat{y}<0 \rightarrow \hat{y}=-1$

Kernel functions

- The basic kernel functions are known, and these functions can be combined to create a new kernel function.

Definition : $\phi(\vec{x})\cdot\phi(\vec{y}) \stackrel{\text{def}}{=} k(\vec{x},\vec{y})$

$\vec{x}=(x_1,x_2) \xrightarrow{\phi}$

Polynomial:
- $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y})^p \leftarrow \phi(\vec{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)$ (p=2)
- $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y}+c)^p,\ (c>0) \leftarrow \phi(\vec{x})=(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2)$ (p=2, c=1)

Gaussian : $k(\vec{x},\vec{y})=\exp(-\gamma\|\vec{x}-\vec{y}\|^2) \leftarrow \phi(\vec{x})=(z_1,z_2,z_3,\ldots)$ (infinite dimension)

$\gamma=\frac{1}{2\sigma^2},\ (\sigma\neq 0)$

Sigmoid : $k(\vec{x},\vec{y})=\tanh(a\vec{x}\cdot\vec{y}+b) \leftarrow \phi(\vec{x})=(z_1,z_2,z_3,\ldots)$

- This cannot be a kernel function in the strict sense because it does not satisfy Mercer's theorem. Nevertheless, it is widely used.
- Hsuan-Tien Lin, 2003, A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods

- proof of $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y})^2$

$\vec{x}=(x_1,x_2),\ \vec{y}=(y_1,y_2),\ p=2$

$k(\vec{x},\vec{y})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)\cdot(y_1^2,\sqrt{2}y_1y_2,y_2^2)$

$= x_1^2y_1^2+2x_1y_1x_2y_2+x_2^2y_2^2$

$= (x_1y_1+x_2y_2)^2=(\vec{x}\cdot\vec{y})^2$

- If k1 and k2 are kernel functions, the functions below that combine them can also be kernel functions.

1) $k(\vec{x},\vec{z})=k_1(\vec{x},\vec{z})+k_2(\vec{x},\vec{z})$
2) $k(\vec{x},\vec{z})=ak_1(\vec{x},\vec{z})$
3) $k(\vec{x},\vec{z})=k_1(\vec{x},\vec{z})k_2(\vec{x},\vec{z})$
4) $k(\vec{x},\vec{z})=f(\vec{x})f(\vec{z})$
5) $k(\vec{x},\vec{z})=k_3(\phi(\vec{x}),\phi(\vec{z}))$
6) $k(\vec{x},\vec{z})=\vec{x}'B\vec{z}$

$f(\cdot)$: a real-valued function on X

$\phi: X\rightarrow R^N$ with k_3 a kernel over $R^N\times R^N$

B : a symmetric positive semi-definite n x n matrix.

- For more detailed information, including proof of this part, please refer to the book below.

Kernel Methods for Pattern Analysis
by John Shawe-Taylor, Nello Cristianini
Chapter 2 : Kernel method : an overview
Chapter 3 : Properties for kernels

- The basic kernel functions are known, and these functions can be combined to create a new kernel function.

Definition : $\phi(\vec{x})\cdot\phi(\vec{y}) \stackrel{\text{def}}{=} k(\vec{x},\vec{y})$

$\vec{x}=(x_1,x_2) \xrightarrow{\phi}$

Polynomial:
- $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y})^p \leftarrow \phi(\vec{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)$ (p=2)
- $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y}+c)^p,\ (c>0) \leftarrow \phi(\vec{x})=(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2)$ (p=2, c=1)

Gaussian : $k(\vec{x},\vec{y})=\exp(-\gamma\|\vec{x}-\vec{y}\|^2) \leftarrow \phi(\vec{x})=(z_1,z_2,z_3,\ldots)$ (infinite dimension)

$\gamma=\frac{1}{2\sigma^2},\ (\sigma\neq 0)$

Sigmoid : $k(\vec{x},\vec{y})=\tanh(a\vec{x}\cdot\vec{y}+b) \leftarrow \phi(\vec{x})=(z_1,z_2,z_3,\ldots)$

- This cannot be a kernel function in the strict sense because it does not satisfy Mercer's theorem. Nevertheless, it is widely used.
- Hsuan-Tien Lin, 2003, A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods

- proof of $k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y})^2$

$\vec{x}=(x_1,x_2),\ \vec{y}=(y_1,y_2),\ p=2$

$k(\vec{x},\vec{y})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)\cdot(y_1^2,\sqrt{2}y_1y_2,y_2^2)$

$= x_1^2y_1^2+2x_1y_1x_2y_2+x_2^2y_2^2$

$= (x_1y_1+x_2y_2)^2=(\vec{x}\cdot\vec{y})^2$

- If k1 and k2 are kernel functions, the functions below that combine them can also be kernel functions.

1) $k(\vec{x},\vec{z})=k_1(\vec{x},\vec{z})+k_2(\vec{x},\vec{z})$
2) $k(\vec{x},\vec{z})=ak_1(\vec{x},\vec{z})$
3) $k(\vec{x},\vec{z})=k_1(\vec{x},\vec{z})k_2(\vec{x},\vec{z})$
4) $k(\vec{x},\vec{z})=f(\vec{x})f(\vec{z})$
5) $k(\vec{x},\vec{z})=k_3(\phi(\vec{x}),\phi(\vec{z}))$
6) $k(\vec{x},\vec{z})=\vec{x}'B\vec{z}$

$f(\cdot)$: a real-valued function on X

$\phi: X\rightarrow R^N$ with k_3 a kernel over $R^N\times R^N$

B : a symmetric positive semi-definite n x n matrix.

- For more detailed information, including proof of this part, please refer to the book below.

Kernel Methods for Pattern Analysis
by John Shawe-Taylor, Nello Cristianini
Chapter 2 : Kernel method : an overview
Chapter 3 : Properties for kernels

- Training data: $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})$ (i = 1, 2, 3, ..., n)
- kernel function : $k(\vec{x}^{(i)}, \vec{x}^{(j)}) = \phi(\vec{x}^{(i)}) \cdot \phi(\vec{x}^{(j)})$
- kernel matrix: K is symmetric matrix.

$$K = \begin{bmatrix} k(x^{(1)}, x^{(1)}) & k(x^{(1)}, x^{(2)}) \\ k(x^{(2)}, x^{(1)}) & k(x^{(2)}, x^{(2)}) \end{bmatrix} \quad k(\vec{x}^{(i)}, \vec{x}^{(j)}) = k(\vec{x}^{(j)}, \vec{x}^{(i)})$$

if n=2.

- Key theorems about symmetric matrices, PSD, and PD.

For a real symmetric matrix M (n x n):
- For any real x vector (n x 1), M is PSD if $x^T \cdot M \cdot x \geq 0$.
- For any real x vector (n x 1), M is PD if $x^T \cdot M \cdot x > 0$.
- If K(.) is a kernel function, K is a symmetric matrix and PSD.
- All eigenvalues of a symmetric matrix M are real numbers.
- If all eigenvalues of a symmetric matrix M are positive, then M is PD.
- If all eigenvalues of a symmetric matrix M are non-negative, then M is PSD.

PSD : Positive Semi-Definite, PD : Positive Definite

- **Mercer's theorem**
  - For training data x: (finite set) – discrete version

$$x^T \cdot K \cdot x \geq 0$$

K : Kernel matrix, symmetric and PSD

$$\sum_i \sum_j K_{i,j} x_i x_j \geq 0$$

  - Continuous version

$$\iint_{x,x'} K(x,x') g(x) g(x') \, dx \, dx' \geq 0$$

* If the above inequality holds, then K is a valid kernel.

- For more detailed information, including proof of this part, please refer to the book below.

Kernel Methods for Pattern Analysis (by John Shawe-Taylor, Nello Cristianini) Chapter 2, 3

Quadratic Programming for nonlinear SVM

- The solution to the Lagrange dual function can be obtained in the same way as linear soft margin SVM.

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y^{(i)} y^{(j)} \underbrace{\phi(\vec{x}^{(i)}) \cdot \phi(\vec{x}^{(j)})}_{\vec{x}^{(i)} \cdot \vec{x}^{(j)}}$$

$$H_{i,j} = y^{(i)} y^{(j)} \phi(\vec{x}^{(i)}) \cdot \phi(\vec{x}^{(j)}) = y^{(i)} y^{(j)} k(\vec{x}^{(i)}, \vec{x}^{(j)}) \longleftarrow \text{definition}$$

$$H = \begin{bmatrix} y^{(1)} y^{(1)} & y^{(1)} y^{(2)} \\ y^{(2)} y^{(1)} & y^{(2)} y^{(2)} \end{bmatrix} \times \begin{bmatrix} k(\vec{x}^{(1)}, \vec{x}^{(1)}) & k(\vec{x}^{(1)}, \vec{x}^{(2)}) \\ k(\vec{x}^{(2)}, \vec{x}^{(1)}) & k(\vec{x}^{(2)}, \vec{x}^{(2)}) \end{bmatrix}$$

← for N=2 element wise product

Kernel matrix (K)

H = np.outer(y, y) * K ← Python code

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j H_{i,j}$$

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} [\lambda_1 \ \lambda_2] \cdot H \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$$

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \lambda^T \cdot H \cdot \lambda$$

$$\text{s.t } 0 \leq \lambda_i \leq C$$

$$\sum_{i=1}^{N} \lambda_i y^{(i)} = 0$$

- Standard form of QP

$$\underset{x}{argmin} \ \frac{1}{2} x^T P x + q^T x$$

$$\text{s.t } Gx \leq h, \quad Ax = b$$

$$Ax = b$$

- Lagrange dual function for nonlinear SVM

$$\underset{\lambda}{argmin} \ \frac{1}{2} \lambda^T H \lambda - 1^T \lambda_i \longleftarrow \underset{\lambda}{argmax} \ L_D \rightarrow \underset{\lambda}{argmin} \ (-L_D)$$

$$\text{s.t } -\lambda_i \leq 0$$

$$\lambda_i \leq C$$

$$\sum_{i=1}^{N} \lambda_i y^{(i)} = 0$$

← Expressed in the same format as the standard form of QP.

$P := H$ size = N x N

$q := -\vec{1}$ = [[-1], [-1], …] - size = N x 1

$A := y$ size = N x 1

$b := 0$ scalar

$$G \cdot \lambda \leq h \longrightarrow \underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{G} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} -\lambda_1 \\ -\lambda_2 \\ -\lambda_3 \\ -\lambda_4 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ C \\ C \\ C \\ C \end{bmatrix}}_{h}$$

$-\lambda_i \leq 0$

$\lambda_i \leq C$