ML&MEA (2024)
Lecture 9 - SVM kernel method

Quanying Liu

BME, SUSTech

2024.3.26

Content

Non-separable Cases

Recall: Two classes of data are **linearly separable**, if and only if there exists a hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ that separates two classes.

In reality, there are non-separable cases. How to classify these two classes?

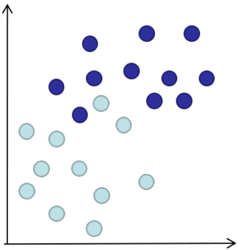
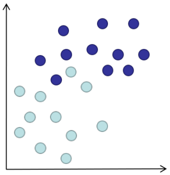


图 1: Non-separable cases

Soft Margin & Kernel Method

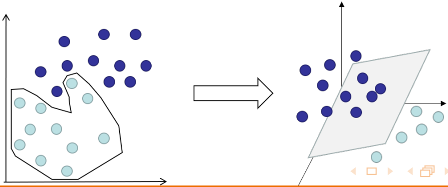
1. Find the optimal hyperplane to minimize classification error with some tolerance (Soft Margin).

(A)



2. Transform data into a higher dimension space where two classes are separable (Kernel Method).

(B)



SVM Soft Margin: the non-negative slack variable ξ_i

Introduce an **non-negative slack variable** ξ_i for $i = \{1, \ldots, N\}$ that satisfies:

1. $\xi_i = 0$: Data point i is not inside the margin of separation
2. $0 \leq \xi_i \leq 1$: Data point is inside the margin of separation and on the correct side of the hyperplane (e.g., ξ_1 and ξ_2)
3. $\xi_i > 1$: Data point is inside the margin of separation but on the wrong side of the hyperplane (e.g., ξ_3 and ξ_4)

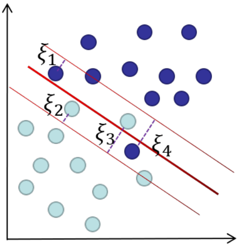
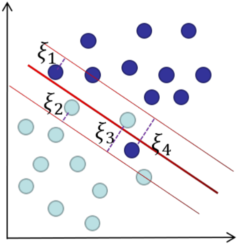


图 2: Slack variable ξ_i

SVM Soft Margin: the new loss function

The optimal hyperplane must penalize the classification error $\sum_{i=1}^{N} \xi_i$.
The new loss function $\mathcal{L}(\mathbf{w}, \boldsymbol{\xi})$ is:

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda \sum_{i=1}^{N} \xi_i$$

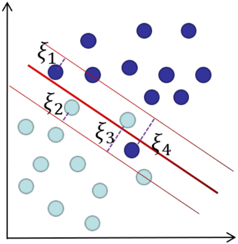


- λ is a hyperparameter, reflecting the cost of violating the margin constraints.
  1. A large λ generally leads to a smaller margin but also fewer misclassification of training data
  2. A small λ generally leads to a larger margin but more misclassification of training data
- As a design parameter, λ can be set by user with $\lambda > 0$.

SVM Soft Margin: the new constraints

Given a data $\mathbf{x}_i$, we have

$$d_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i$$



If $\xi_i = 0$, then the constraint is the same as that in basic version (Hard Margin).

SVM with soft margin:

Primal problem:

$$\begin{aligned} &\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\sum_{i=1}^{N}\xi_i \\ &s.t.\ \ d_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) - 1 + \xi_i \geq 0 \\ &\qquad \xi_i \geq 0 \end{aligned} \tag{1}$$

SVM Soft Margin: the dual problem

Let α_i and β_i be the Lagrange multipliers for the two constraints.

The **generalized Lagrangian function** is: $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

$$
\begin{aligned}
&= \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(d_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{N}\beta_i\xi_i \\
&= \frac{\mathbf{w}^T\mathbf{w}}{2} + \lambda\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i d_i\mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{N}\alpha_i d_i + \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}\alpha_i\xi_i - \sum_{i=1}^{N}\beta_i\xi_i
\end{aligned}
$$

The constrained Primal problem becomes the **dual problem**:

$$
\begin{aligned}
\min_{\mathbf{w}, b, \xi}\max_{\alpha, \beta} \quad & \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
s.t. \quad & \alpha_i \geq 0, \quad \beta_i \geq 0
\end{aligned} \tag{2}
$$

Solving the dual problem: Step 1

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{N}\alpha_i d_i + \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}\alpha_i\xi_i - \sum_{i=1}^{N}\beta_i\xi_i$$

We have derived:

$\partial\mathcal{L}/\partial\mathbf{w} = \mathbf{w} - \sum_{i=1}^{N}\alpha_i d_i \mathbf{x}_i = \mathbf{0} \quad \Longrightarrow \mathbf{w} = \sum_{i=1}^{N}\alpha_i d_i \mathbf{x}_i$

$\partial\mathcal{L}/\partial b = -\sum_{i=1}^{N}\alpha_i d_i = 0 \quad \Longrightarrow \sum_{i=1}^{N}\alpha_i d_i = 0$

$\partial\mathcal{L}/\partial\xi_i = \lambda - \alpha_i - \beta_i = 0 \quad \Longrightarrow \lambda = \alpha_i + \beta_i$

$$\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i d_i \alpha_j d_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\sum_{i=1}^{N}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i = \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i d_i \alpha_j d_j \mathbf{x}_i^T\mathbf{x}_j$$

Solving the dual problem: Step 2

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^{N} \alpha_i d_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i - \sum_{i=1}^{N} \beta_i \xi_i$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T \mathbf{x}_i + \lambda \sum_{i=1}^{N} \xi_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i - \sum_{i=1}^{N} \beta_i \xi_i$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i d_i \alpha_j d_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{N} \underbrace{(\alpha_i + \beta_i)}_{\lambda} \xi_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i - \sum_{i=1}^{N} \beta_i \xi_i$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i d_i \alpha_j d_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{N} \alpha_i \quad \triangleq Q(\boldsymbol{\alpha})$$

Solving the dual problem: Step 3

Formulate the Dual Problem (with soft margin):

$$\text{Maximize}_{\boldsymbol{\alpha}} : \quad Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{Subject to:} \quad \sum_{i=1}^{N} \alpha_i d_i = 0 \quad \text{and} \quad 0 \le \alpha_i \le \lambda$$

- Note: $Q(\boldsymbol{\alpha})$ is same as the dual problem without soft margin.
- The effect of the error penalty $\sum_{i=1}^{N} \xi_i$ on the optimization problem is to set an upper bound for Lagrange multipliers α_i.
- In KKT conditions, $\lambda = \alpha_i + \beta_i$ with $\alpha_i \ge 0$ and $\beta_i \ge 0$.
  $\Longrightarrow 0 \le \alpha_i \le \lambda$.

Solving α^* with Quadratic programming

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{N}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i d_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq \lambda$$

Solving with **Linear Quadratic Solver**:

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T \begin{bmatrix} d_1 d_1 \mathbf{x}_1^T\mathbf{x}_1 & d_1 d_2 \mathbf{x}_1^T\mathbf{x}_2 & \cdots & d_1 d_n \mathbf{x}_1^T\mathbf{x}_n \\ d_2 d_1 \mathbf{x}_2^T\mathbf{x}_1 & d_2 d_2 \mathbf{x}_2^T\mathbf{x}_2 & \cdots & d_2 d_n \mathbf{x}_2^T\mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 \mathbf{x}_n^T\mathbf{x}_1 & d_n d_2 \mathbf{x}_n^T\mathbf{x}_2 & \cdots & d_n d_n \mathbf{x}_n^T\mathbf{x}_n \end{bmatrix} \boldsymbol{\alpha} + (-\mathbf{1})^T\boldsymbol{\alpha}$$

$$\text{s.t.} \quad \mathbf{d}^T\boldsymbol{\alpha} = 0 \quad \text{and} \quad \mathbf{0} \leq \boldsymbol{\alpha} \leq [\lambda, \lambda, \ldots, \lambda]^T$$

Solving SVM with Hinge loss

Hinge loss: $\mathcal{L}_i = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$

gradient: $\nabla_{\mathbf{w}} \mathcal{L} = \lambda \mathbf{w} - y_i \mathbf{x}_i, \quad \nabla_b \mathcal{L} = -y_i, \text{if } 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0$

$\nabla_{\mathbf{w}} \mathcal{L} = \lambda \mathbf{w}, \quad \nabla_b \mathcal{L} = 0, \text{if } 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0$

Solving with **Gradient Descent**:

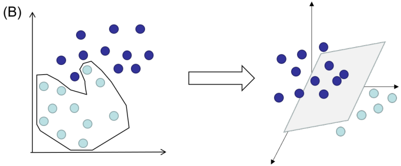
```
for epoch in range(epochs):
for i, x in enumerate(X):
Calculate the condition for the hinge loss
condition = y[i] * (np.dot(x, weights) + bias)

Check if the data point is misclassified or within the margin
if condition < 1:
Update for misclassified points or points within the margin
weights -= learning_rate * (2 * lambda_param * weights - y[i] * x)
bias -= learning_rate * (-y[i])
else:
Update only based on the regularizer for correctly classified points outside the margin
weights -= learning_rate * (2 * lambda_param * weights)
```

Cover's Theorem

Cover's Theorem: [by Thomas M. Cover, in 1965]

> A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.



Given a set of training data that is not linearly separable, one can with high probability transform it into a training set that is linearly separable by projecting it into a higher-dimensional space via some non-linear transformation.

Transformation into a higher-D space

Given a **transformation** $\varphi()$, we search for an optimal **hyperplane** in higher-D feature space to separate the two classes,

$$g(\mathbf{x}) = {\mathbf{w}^*}^T \varphi(\mathbf{x}) + b^* = 0 \tag{3}$$

For the training data $\mathbf{x}_i$, using the sign function (符号函数) for classification as

$$\begin{cases} \text{sign}(\mathbf{x}_i) = {\mathbf{w}^*}^T \varphi(\mathbf{x}_i) + b^* \geq +1 & \text{for } d_i = +1 \\ \text{sign}(\mathbf{x}_i) = {\mathbf{w}^*}^T \varphi(\mathbf{x}_i) + b^* \leq -1 & \text{for } d_i = -1 \end{cases}$$

or, in a compact form:

$$d_i({\mathbf{w}^*}^T \varphi(\mathbf{x}_i) + b^*) \geq 1 \tag{4}$$

Q: How to design a good $\varphi()$, and to find ${\mathbf{w}^*}^T$ and b^*?

Dual problem in Kernel Method

Recall the generalized Lagrangian function for SVM Hard Margin,

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i \left(d_i \left(\mathbf{w}^T\mathbf{x}_i + b\right) - 1\right)$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{N} \alpha_i d_i + \sum_{i=1}^{N} \alpha_i$$

The Dual Problem for SVM Kernel Method:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i \left(d_i \left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right) + b\right) - 1\right)$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T\varphi\left(\mathbf{x}_i\right) - b\sum_{i=1}^{N} \alpha_i d_i + \sum_{i=1}^{N} \alpha_i$$

Dual problem in Kernel Method

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T \varphi(\mathbf{x}_i) - b\sum_{i=1}^{N}\alpha_i d_i + \sum_{i=1}^{N}\alpha_i$$

From KKT conditions, we have $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$ and $\frac{\partial \mathcal{L}}{\partial b} = 0$.
Then, we can derive:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \varphi(\mathbf{x}_i) \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i d_i = 0$$

We can further derive

$$\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T \varphi(\mathbf{x}_i) = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j)$$

Dual problem in Kernel Method

We define $Q(\boldsymbol{\alpha})$ as follows,

$$Q(\boldsymbol{\alpha}) \triangleq \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \underbrace{\varphi^T(\mathbf{x}_i) \varphi(\mathbf{x}_j)}_{K(\mathbf{x}_i, \mathbf{x}_j)}$$

The **Kernel**, $K(\mathbf{x}_i, \mathbf{x}_j)$, is defined as

$$\underbrace{K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)}_{\text{symmetric}} = \varphi^T(\mathbf{x}_i) \varphi(\mathbf{x}_j) = \varphi^T(\mathbf{x}_j) \varphi(\mathbf{x}_i)$$

Example: $\mathbf{x} = [x_1, x_2]^T \longrightarrow$ 2nd-order φ

$$\varphi(\mathbf{x}) = [1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ x_2^2,\ x_1 x_2]^T$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = ?$$

Dual problem in Kernel Method

We define $Q(\boldsymbol{\alpha})$ as follows,

$$Q(\boldsymbol{\alpha}) \triangleq \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \underbrace{\varphi^T(\mathbf{x}_i) \varphi(\mathbf{x}_j)}_{K(\mathbf{x}_i, \mathbf{x}_j)}$$

The **Kernel**, $K(\mathbf{x}_i, \mathbf{x}_j)$, is defined as

$$\underbrace{K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)}_{\text{symmetric}} = \varphi^T(\mathbf{x}_i)\, \varphi(\mathbf{x}_j) = \varphi^T(\mathbf{x}_j)\, \varphi(\mathbf{x}_i)$$

Example: $\mathbf{x} = [x_1, x_2]^T \longrightarrow$ 2nd-order φ

$$\varphi(\mathbf{x}) = [1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ x_2^2,\ x_1 x_2]^T$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = 1 + 2x_{i,1}x_{j,1} + 2x_{i,2}x_{j,2} + x_{i,1}^2 x_{j,1}^2 + x_{i,2}^2 x_{j,2}^2 + 2x_{i,1}x_{j,1}x_{i,2}x_{j,2}$$

Soft Margin + transformation

Dual Problem with Soft Margin

$$\text{Maximize: } Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{Subject to: } \sum_{i=1}^{N} \alpha_i d_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq \lambda$$

Dual Problem with Soft Margin and Transformation

$$\text{Maximize: } Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j)$$

$$\text{Subject to: } \sum_{i=1}^{N} \alpha_i d_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq \lambda$$

Classification in transformation space

If a data point $\mathbf{x}^{(sv)}$ is a Support Vector with its label $d^{(vs)}$, considering the solution of Lagrangian multipliers, α_i^*,

$$\begin{cases} \mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* d_i \varphi(\mathbf{x}_i) \\ b^* = \frac{1}{d^{(sv)}} - \mathbf{w}^{*T} \varphi\left(\mathbf{x}^{(sv)}\right) \end{cases}$$

Discriminant function $g(\mathbf{x})$ is defined as

$$g(\mathbf{x}) = \mathbf{w}^{*T} \varphi(\mathbf{x}) + b^* = \sum_{i=1}^{N} \alpha_i^* d_i \underbrace{\varphi^T(\mathbf{x}_i) \varphi(\mathbf{x})}_{K(\mathbf{x}_i, \mathbf{x})} + b^*$$

To classify a new data point $\mathbf{x}_{\text{new}}$

$$d_{\text{new}} = \operatorname{sgn}\left[g\left(\mathbf{x}_{\text{new}}\right)\right]$$

Main issue: We need design $\varphi(\cdot)$

Kernel trick: Design the expression for $K(\cdot,\cdot)$ directly.

Kernel trick

Kernel trick: Design the expression for $K(\cdot,\cdot)$ directly. Example:
$\mathbf{x} = [x_1, x_2]^T \longrightarrow$ 2nd-order φ

$$\varphi(\mathbf{x}) = [1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ x_2^2,\ \sqrt{2}x_1x_2]^T$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = 1 + 2x_{i,1}x_{j,1} + 2x_{i,2}x_{j,2} + x_{i,1}^2x_{j,1}^2 + x_{i,2}^2x_{j,2}^2 + 2x_{i,1}x_{j,1}x_{i,2}x_{j,2}$$

Now, Let us design the kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ directly.

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T\mathbf{x}_j)^2 \\ &= (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2})^2 \\ &= 1 + 2x_{i,1}x_{j,1} + 2x_{i,2}x_{j,2} + x_{i,1}^2x_{j,1}^2 + x_{i,2}^2x_{j,2}^2 + 2x_{i,1}x_{j,1}x_{i,2}x_{j,2} \end{aligned}$$

The polynomial kernel

- The data $\mathbf{x} = [x_1,\ x_2,\ \dots,\ x_p]^T \in \mathbb{R}^p$

The polynomial kernel

- The data $\mathbf{x} = [x_1, \ x_2, \ \dots, \ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.

The polynomial kernel

- The data $\mathbf{x} = [x_1,\ x_2,\ \ldots,\ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.
- The 'equivalent' kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T \mathbf{x})^q$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \cdots + x_{i,p}x_{j,p})^q$$

The polynomial kernel

- The data $\mathbf{x} = [x_1, \ x_2, \ \ldots, \ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.
- The 'equivalent' kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T\mathbf{x})^q$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \cdots + x_{i,p}x_{j,p})^q$$

- Q: Why can the kernel trick help us, compared with a nonlinear transformation of $\mathbf{x}$?

The polynomial kernel

- The data $\mathbf{x} = [x_1,\ x_2,\ \dots,\ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.
- The 'equivalent' kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T\mathbf{x})^q$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \cdots + x_{i,p}x_{j,p})^q$$

- Q: Why can the kernel trick help us, compared with a nonlinear transformation of $\mathbf{x}$?

Consider the situation:

Dimension of data: $p = 10$

Polynomial order: $q = 100$

The polynomial kernel

- The data $\mathbf{x} = [x_1, \ x_2, \ \ldots, \ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.
- The 'equivalent' kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T \mathbf{x})^q$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \cdots + x_{i,p}x_{j,p})^q$$

- Q: Why can the kernel trick help us, compared with a nonlinear transformation of $\mathbf{x}$?

  Consider the situation:
  Dimension of data: $p = 10$
  Polynomial order: $q = 100$

- We can define any function $K(\mathbf{x}_i, \mathbf{x}_j)$, but it should correspond to a nonlinear transformation $\varphi(\mathbf{x})$.
  $\Longrightarrow K(\mathbf{x}_i, \mathbf{x}_j) = \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j)$

The polynomial kernel

- The data $\mathbf{x} = [x_1,\ x_2,\ \ldots,\ x_p]^T \in \mathbb{R}^p$
- **The polynomial function:** $\varphi(\mathbf{x})$ is polynomial of order q.
- The 'equivalent' kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T\mathbf{x})^q$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + \cdots + x_{i,p}x_{j,p})^q$$

- Q: Why can the kernel trick help us, compared with a nonlinear transformation of $\mathbf{x}$?

  Consider the situation:
  Dimension of data: $p = 10$
  Polynomial order: $q = 100$

- We can define any function $K(\mathbf{x}_i, \mathbf{x}_j)$, but it should correspond to a nonlinear transformation $\varphi(\mathbf{x})$.
  $\Longrightarrow K(\mathbf{x}_i, \mathbf{x}_j) = \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j)$
- Q: what kind of $K(\cdot, \cdot)$ is a valid kernel?

The requirement to design a kernel $K(\cdot,\cdot)$

For training set $S=\{(x_i,d_i)\}_{i=1}^N$, the Gram matrix K is

$$\mathbf{K}=\left[\begin{array}{ccc} K\left(\mathbf{x}_1,\mathbf{x}_1\right) & \ldots & K\left(\mathbf{x}_1,\mathbf{x}_N\right) \\ \vdots & \ddots & \vdots \\ K\left(\mathbf{x}_N,\mathbf{x}_1\right) & \ldots & K\left(\mathbf{x}_N,\mathbf{x}_N\right) \end{array}\right] \in R^{N\times N}$$

Mercer's condition:
Gram matrix K is positive semi-definite (i.e., its eigenvalues are nonnegative).

A kernel satisfying the Mercer condition ensures the existence of a global optimum for the resulting optimization problem.

Eigenvalue and Eigenvector

For a square matrix $A \in \mathbb{R}^{n \times n}$, its eigenvalue λ and eigenvector $\mathbf{x}$:

$$A\mathbf{x} = \lambda \mathbf{x}$$

For all eigenvalues and eigenvectors, we can derive

$$\begin{aligned} A\left[\mathbf{x}_1 \ \mathbf{x}_2 \ \ldots \ \mathbf{x}_n\right] &= \left[\lambda_1 \mathbf{x}_1 \ \lambda_2 \mathbf{x}_2 \ \ldots \ \lambda_n \mathbf{x}_n\right] \\ &= \left[\mathbf{x}_1 \ \mathbf{x}_2 \ \ldots \ \mathbf{x}_n\right] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \end{aligned}$$

If the n eigenvalues exist, the eigendecomposition of A is

$$A = S \Lambda S^{-1} \tag{5}$$

Watch Gilbert Strange 22:
`https://www.bilibili.com/video/BV1zx411g7gq?p=22`

Positive Definite Matrix (正定矩阵)

- For a **symmetric** matrix $A \in \mathbb{R}^{n \times n}$, the eigendecomposition

$$A = Q \Lambda Q^{-1},$$

Q is orthnormal $\Longrightarrow QQ^T = \mathbf{I}$ and $\|Q\| = 1$
and all eigenvalues λ_i are **real** (no complex part).

- Definition of **Positive Definite** matrix:
If $A \in \mathbb{R}^{n \times n}$ is Positive Definite, then $X^T AX > 0$ for any matrix X. The eigendecomposition:

$$A = Q \Lambda Q^{-1},$$

Q is orthnormal $\Longrightarrow QQ^T = \mathbf{I}$ and $\|Q\| = 1$
and all eigenvalues λ_i are **positive** $(\lambda_i > 0)$.

- Positive **semi**-Definite matrix?

Watch Gilbert Strange 26:
`https://www.bilibili.com/video/BV1zx411g7gq?p=26`

Basic Kernels 1

The ith data: $\mathbf{x}_i = [x_1, \ x_2, \ \ldots, \ x_p]^T \in \mathbb{R}^p$

- **Linear Kernel** 线性核函数：主要用于线性可分的情形。参数少，速度快，对于一般数据，分类效果已经很理想了。

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j + c$$

- **Polynomial Kernel** 多项式核函数:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^q$$

- **Gaussian kernel** 高斯核函数:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$$

Basic Kernels 2

- **Radial Basis Function (RBF) kernel** 径向基核函数：主要用于线性不可分的情形。How to choose hyperparameter γ?

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$$

- **Laplace RBF kernel**:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\sigma})$$

- **Sigmoid kernel** :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i^T \mathbf{x}_j + c)^q$$

Choose kernel and tune hyperparameters (See Q&A)

Kernel 的选择，吴恩达的建议如下：

- 如果 Feature 的数量很大，跟样本数量差不多，这时候选用 LR 或者是 Linear Kernel 的 SVM
- 如果 Feature 的数量比较小，样本数量一般，不算大也不算小，选用 SVM+Gaussian Kernel
- 如果 Feature 的数量比较小，而样本数量很多，需要手工添加一些 feature 变成第一种情况

Hyperparameters 的选择

- 验证曲线法 validation curve: `https://www.jianshu.com/p/6d4b7f3b7c14`
- 网格搜索法 grid search: `https://www.jianshu.com/p/7701eab3bbc9`

Build a SVM with Kernel trick

Given a training set $S = \{(x_i, d_i)\}_{i=1}^{N}$

- Design a suitable kernel and check Mercer's condition
- Choose a value for the hyperparameter λ
- Solve α_i^*
- Determine b^* in

$$g(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i^* d_i K(\mathbf{x}, \mathbf{x}_i) + b^*$$

using the fact that for a support vector $\mathbf{x}^{(sv)}$

$$g\left(\mathbf{x}^{(sv)}\right) = \pm 1 = d^{(sv)}$$

For a valid kernel $K(\cdot,\cdot)$, the Gram matrix is positive semi-definite

For training set $S = \{(x_i, d_i)\}_{i=1}^{N}$, the Gram matrix K is

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \ldots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \ldots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \in R^{N\times N}$$

Mercer's condition:
Gram matrix K is positive semi-definite (i.e., its eigenvalues are nonnegative).

Proof: Mercer's condition $\Longrightarrow K(\mathbf{x}_i, \mathbf{x}_j) = \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j)$

Watch 机器学习白板推导 38 for the proof:
https://www.bilibili.com/video/BV1aE411o7qd?p=38