Support Vector Machine
Outline

- Hinge Loss
- Kernel Method

Support Vector Machine (SVM)
Binary Classification

• Step 1: Function set (Model)

\[ g(x) = \begin{cases} 
  f(x) > 0 & \text{Output} = +1 \\
  f(x) < 0 & \text{Output} = -1 
\end{cases} \]

• Step 2: Loss function:

\[ L(f) = \sum_n \delta(g(x^n) \neq \hat{y}^n) \cdot l(f(x^n), \hat{y}^n) \]

The number of times \( g \) get incorrect results on training data.

• Step 3: Training by gradient descent is difficult

Gradient descent is possible if \( g(*) \) and \( \delta(*) \) is differentiable
**Step 2: Loss function**

\[ g(x) = \begin{cases} 
  +1 & f(x) > 0 \\
  -1 & f(x) < 0 
\end{cases} \]

- Ideal loss:
  \[ L(f) = \sum_n \delta(g(x^n) \neq \hat{y}^n) \]

- Approximation:
  \[ L(f) = \sum_n l(f(x^n), \hat{y}^n) \]

Ideal loss: \( \delta(g(x^n) \neq \hat{y}^n) \)  

Larger value, smaller loss  

\( \hat{y}^n f(x) \)
**Step 2: Loss function**

Square Loss:

- If $\hat{y}^n = 1$, $f(x)$ close to 1
- If $\hat{y}^n = -1$, $f(x)$ close to -1

\[
\ell(f(x^n), \hat{y}^n) = (\hat{y}^n f(x^n) - 1)^2
\]

- Ideal loss: Larger value, smaller loss
- Square loss

不合理 ...

Larger value, smaller loss

$\hat{y}^n f(x)$
**Step 2: Loss function**

Sigmoid + Square Loss:

\[
l(f(x^n), \hat{y}^n) = (\sigma(\hat{y}^n f(x)) - 1)^2
\]

If \( \hat{y}^n = 1 \), \( \sigma(f(x)) \) close to 1

If \( \hat{y}^n = -1 \), \( \sigma(f(x)) \) close to 0

\[
(\sigma(f(x)) - 1)^2
\]

\[
(1 - \sigma(f(x)) - 1)^2
\]

\[
(\sigma(f(x)))^2
\]

Square loss

Ideal loss

Larger value, smaller loss
Step 2: Loss function

Sigmoid + cross entropy (logistic regression)

\[
\hat{y}^n = +1, \quad \sigma(f(x))
\]

\[
\hat{y}^n = -1, \quad 1 - \sigma(f(x))
\]

Ideal loss

\[
l(f(x^n), \hat{y}^n) = \ln \left( 1 + \exp \left( -\hat{y}^n f(x) \right) \right)
\]

Square loss

Larger value, smaller loss

Divided by \ln 2 here

努力可以有回報

沒有回報不想努力
**Step 2: Loss function**

\[ l(f(x^n), \hat{y}^n) = \max(0, 1 - \hat{y}^n f(x)) \]

If \( \hat{y}^n = 1 \),
\[ \max(0, 1 - f(x)) \]

If \( \hat{y}^n = -1 \),
\[ \max(0, 1 + f(x)) \]

- **Ideal loss**
- **Square loss**
- **Sigmoid + Square loss**
- **Sigmoid + cross entropy**
- **Hinge Loss**

**Graphical Representation:**
- **Ideal loss**: Horizontal line at 1.
- **Square loss**: Quadratic function.
- **Sigmoid + Square loss**: Combination of sigmoid and square loss.
- **Sigmoid + cross entropy**: Combination of sigmoid and cross-entropy loss.
- **Hinge Loss**: Linear function.

**Points:**
- **Penalty**
- **Good enough**

**Equations:**
- If \( \hat{y}^n = 1 \),
  \[ 1 - f(x) < 0 \quad f(x) > 1 \]
- If \( \hat{y}^n = -1 \),
  \[ 1 + f(x) < 0 \quad f(x) < -1 \]
Compared with logistic regression, linear SVM has different loss function.

**Step 1: Function (Model)**

\[ f(x) = \sum_i w_i x_i + b = [w] \cdot [x] = w^T x \]

**Step 2: Loss function**

\[ L(f) = \sum_n l(f(x^n), \hat{y}^n) + \lambda \|w\|_2 \]

\[ l(f(x^n), \hat{y}^n) = \max(0, 1 - \hat{y}^n f(x)) \]

**Step 3: gradient descent?**

Recall relu, maxout network

Linear SVM – gradient descent

Ignore regularization for simplicity

\[ L(f) = \sum_n l(f(x^n), \hat{y}^n) \quad l(f(x^n), \hat{y}^n) = \max(0, 1 - \hat{y}^n f(x^n)) \]

\[ \frac{\partial l(f(x^n), \hat{y}^n)}{\partial w_i} = \frac{\partial l(f(x^n), \hat{y}^n)}{\partial f(x^n)} \frac{\partial f(x^n)}{\partial w_i} x_i^n \]

\[ f(x^n) = w^T \cdot x^n \]

\[ \frac{\partial \max(0, 1 - \hat{y}^n f(x^n))}{\partial f(x^n)} = \begin{cases} -\hat{y}^n & \text{if } \hat{y}^n f(x^n) < 1 \\ 0 & \text{otherwise} \end{cases} \]

\[ \frac{\partial L(f)}{\partial w_i} = \sum_n \frac{-\delta(\hat{y}^n f(x^n) < 1)\hat{y}^n x_i}{c^n(w)} \quad w_i \leftarrow w_i - \eta \sum_n c^n(w)x_i^n \]
Linear SVM – another formulation

Minimizing loss function $L$: 

$$L(f) = \sum_{n} \varepsilon^n + \lambda \|w\|_2$$

where 

$$\varepsilon^n = \max(0, 1 - \hat{y}^n f(x))$$

$\varepsilon^n$: slack variable

Quadratic programming problem

$$\varepsilon^n \geq 0$$

$$\varepsilon^n \geq 1 - \hat{y}^n f(x) \Rightarrow \hat{y}^n f(x) \geq 1 - \varepsilon^n$$
Dual Representation

$$w^* = \sum_n \alpha_n^* x^n$$  
Linear combination of data points

$$\alpha_n^*$$ may be sparse  \(\rightarrow\)  \(x^n\) with non-zero \(\alpha_n^*\) are support vectors

$$w_i \leftarrow w_i - \eta \sum_n c^n(w)x_i^n$$

If \(w\) initialized as \(0\)

$$w \leftarrow w - \eta \sum_n c^n(w)x^n$$

$$c^n(w) = \frac{\partial l(f(x^n), y^n)}{\partial f(x^n)}$$  
Hinge loss: usually zero

c.f. for logistic regression, it is always non-zero
Dual Representation

\[ w = \sum_n \alpha_n x^n = X\alpha \]

\[ X = \begin{bmatrix} x^1 & x^2 & \cdots & x^N \end{bmatrix} \]

\[ \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} \]

Step 1: \[ f(x) = w^T x \]

\[ f(x) = \sum_n \alpha_n (x^n \cdot x) \]

\[ = \sum_n \alpha_n K(x^n, x) \]

\[ w = X\alpha \]

\[ f(x) = \alpha^T X^T x \]
Dual Representation

Step 1: \[ f(x) = \sum_{n} \alpha_n K(x^n, x) \]

Step 2, 3: Find \{\alpha_1^*, \ldots, \alpha_n^*, \ldots, \alpha_N^*\}, minimizing loss function \( L \)

\[ L(f) = \sum_{n} l(f(x^n), \hat{y}^n) \]

\[ = \sum_{n} l \left( \sum_{n'} \alpha_{n'} K(x^{n'}, x^n), \hat{y}^n \right) \]

We don’t really need to know vector \( x \)
We only need to know the inner project between a pair of vectors \( x \) and \( z \)

\[ K(x, z) \]

Kernel Trick
Kernel Trick

Directly computing $K(x, z)$ can be faster than “feature transformation + inner product” sometimes.

Kernel trick is useful when we transform all $x$ to $\phi(x)$

$$K(x, z) = \phi(x) \cdot \phi(z) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ \sqrt{2}z_1z_2 \\ z_2^2 \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} = (x_1 z_1 + x_2 z_2)^2 = \left( \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \right)^2 = (x \cdot z)^2$$
Kernel Trick

Directly computing $K(x, z)$ can be faster than “feature transformation + inner product” sometimes.

$$K(x, z) = (x \cdot z)^2$$

$$= (x_1z_1 + x_2z_2 + \cdots + x_kz_k)^2$$

$$= x_1^2z_1^2 + x_2^2z_2^2 + \cdots + x_k^2z_k^2$$

$$+ 2x_1x_2z_1z_2 + 2x_1x_3z_1z_3 + \cdots$$

$$+ 2x_2x_3z_2z_3 + 2x_2x_4z_2z_4 + \cdots$$

$$= \phi(x) \cdot \phi(z)$$

$x = \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}$

$z = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix}$

$\phi(x) = \begin{bmatrix} x_1^2 \\ \vdots \\ x_k^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_2x_3 \\ \vdots \end{bmatrix}$
Radial Basis Function Kernel

\[ K(x, z) = \exp \left( -\frac{1}{2} \| x - z \|_2 \right) = \phi(x) \cdot \phi(z) \]

\[ = \exp \left( -\frac{1}{2} \| x \|_2 - \frac{1}{2} \| z \|_2 + x \cdot z \right) \]

\[ = \exp \left( -\frac{1}{2} \| x \|_2 \right) \exp \left( -\frac{1}{2} \| z \|_2 \right) \exp(x \cdot z) = C_x C_z \exp(x \cdot z) \]

\[ = C_x C_z \sum_{i=0}^{\infty} \frac{(x \cdot z)^i}{i!} = C_x C_z + C_x C_z (x \cdot z) + C_x C_z \frac{1}{2} (x \cdot z)^2 \ldots \]
Sigmoid Kernel \[ K(x, z) = \tanh(x \cdot z) \]

- When using sigmoid kernel, we have a 1 hidden layer network.

\[ f(x) = \sum_n \alpha_n K(x^n, x) = \sum_n \alpha^n \tanh(x^n \cdot x) \]

The weight of each neuron is a data point

The number of support vectors is the number of neurons.
You can directly design $K(x, z)$ instead of considering $\phi(x), \phi(z)$. When $x$ is structured object like sequence, hard to design $\phi(x)$. $K(x, z)$ is something like similarity (Mercer’s theory to check).

Evaluate the similarity between sequences as $K(x, z)$.

More about kernel design in [Bishop chapter 6.2].


SVM related methods

• Support Vector Regression (SVR)
  • [Bishop chapter 7.1.4]
• Ranking SVM
  • [Alpaydin, Chapter 13.11]
• One-class SVM
  • [Alpaydin, Chapter 13.11]
Deep Learning

Feature transformation

$\phi(x)$

linear classifier

$y_1$

$y_2$

$\vdots$

$y_M$

SVM

Based on kernel function

$\phi(x)$

Input Space

Feature Space

Multiple Kernel learning [Alpaydin, Chapter 13.8]