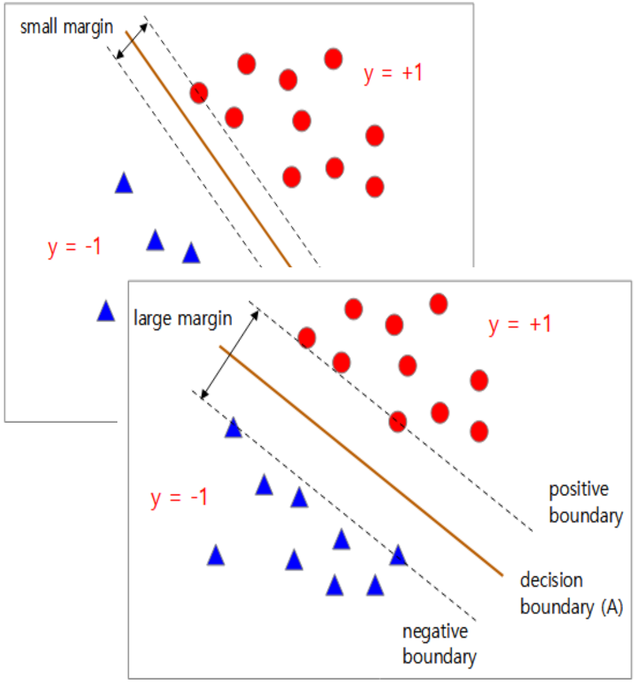


6. Support Vector Machine (SVM)

Part 1: Linear Hard Margin - Algorithm

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

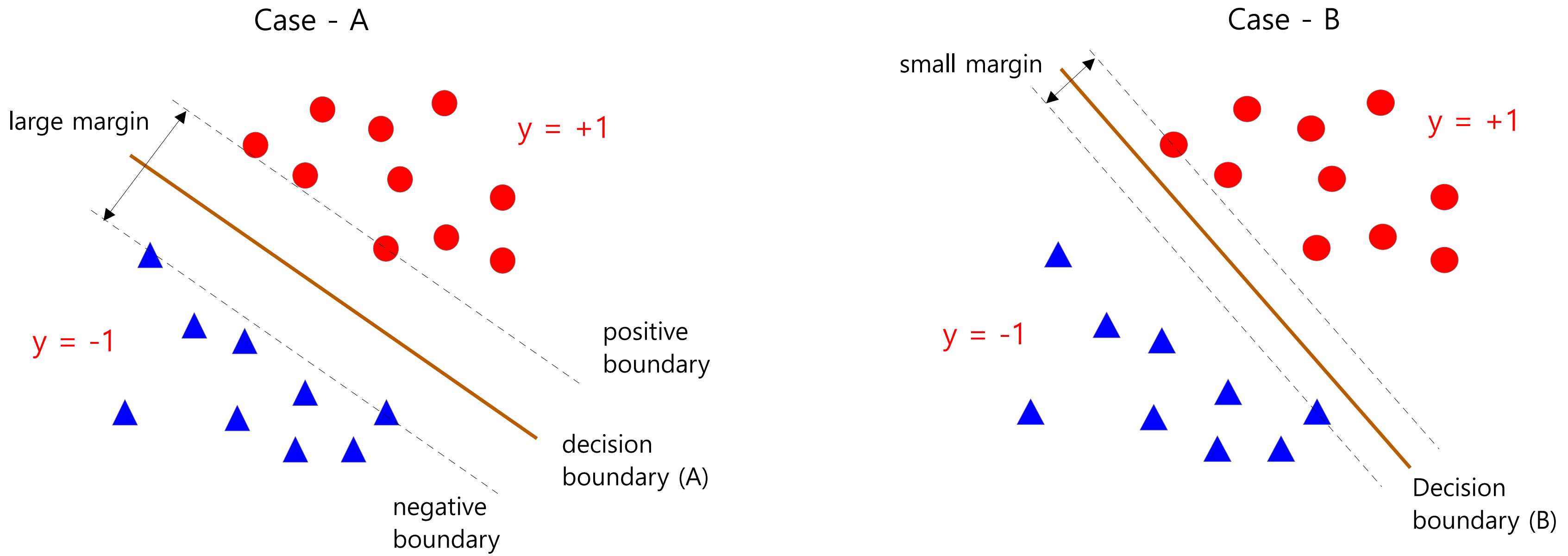
Classification

Regression

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

The best decision boundary

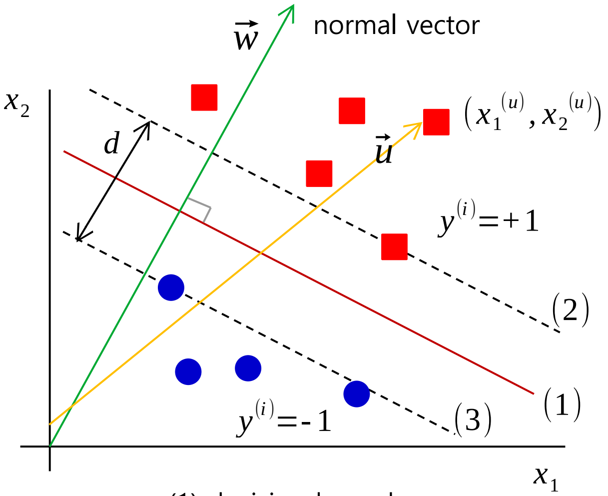
- In the figure below, there are numerous decision boundaries that can distinguish two data clusters.
- Draw an arbitrary straight line between the two clusters and shift the line parallel to the left and right until it reaches the data point, creating two straight lines. The best decision boundary is the one with the largest distance between the two straight lines.
- A support vector machine aim to find the best decision boundary that best separates a dataset into two classes.
- The distance is called the margin. Support vector machine is an algorithm that finds the decision boundary with the largest margin.



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

Linear Hard Margin: Decision rule and Constraints

- SVM is an algorithm that maximizes the distance (d) between the two straight lines (2) and (3) in the figure below.



(1) decision boundary
(2) positive boundary
(3) negative boundary

(1) $w_1x_1 + w_2x_2 + b = 0 \quad \vec{w} = (w_1, w_2)$

- Decision rule

$\vec{w}\cdot\vec{u} \ge c,\ u_{class} = y^{(u)} = +1$ ← If the magnitude of u vector going in the direction of w vector is greater than or equal to a certain number c, that data point is classified as +1.

$\vec{w}\cdot\vec{u} + b \ge 0$ ← Determine $\vec{w}$ and b using the given data.

- constraints

$\vec{w}\cdot\vec{x}^{(+)} + b \ge k$ ← Positive data points are above the positive boundary (2).

$\vec{w}\cdot\vec{x}^{(-)} + b \le -k$ ← Negative data points are below the negative boundary (3).

$\vec{w}'\cdot\vec{x}^{(+)} + b' \ge 1$ ← Divide both sides by k to get standardized w' and b'.

$\vec{w}'\cdot\vec{x}^{(-)} + b' \le -1$ Let's just write w' and b' as w and b.

$y^{(i)}(\vec{w}\cdot\vec{x}^{(i)} + b) \ge 1$ ← $y^{(i)} = \begin{cases} 1 & for\ \ "+" \\ -1 & for\ \ "-" \end{cases}$

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

$y^{(i)}(\vec{w}\cdot\vec{x}^{(i)} + b) - 1 = 0$ ← The data points x on the positive or negative boundary are called support vectors.

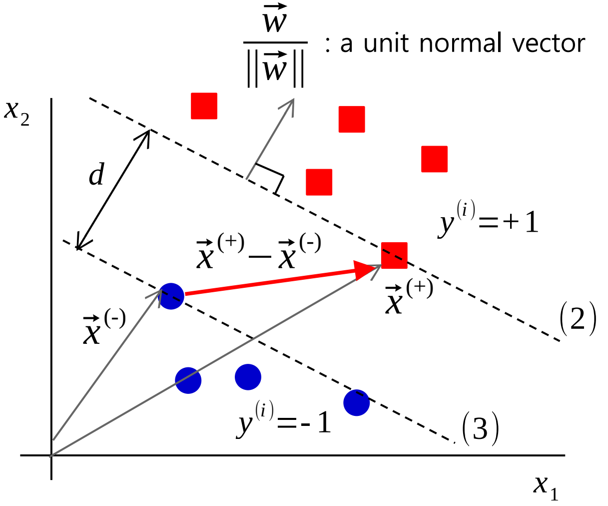
* reference : (MIT lecture) https://www.youtube.com/watch?v=_PwhiWxHK8o

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

Objective function

- Our goal is to maximize the distance between the positive and negative boundaries.

$\frac{\vec{w}}{\|\vec{w}\|}$: a unit normal vector



(2) positive boundary
(3) negative boundary

- Objective function

$$d=(\vec{x}^{(+)}-\vec{x}^{(-)})\cdot\frac{\vec{w}}{\|\vec{w}\|}$$

The distance or margin between the positive and negative boundaries. This is the magnitude of the red vector going in the direction of the unit w vector.

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

Both x+ and x- samples meet this condition because they are support vectors.

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

for x+ samples. y= +1

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

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

for x- samples. y= -1

$$\vec{w}\cdot\vec{x}^{(-)}=-1-b$$

$$d=\frac{(\vec{x}^{(+)}\cdot\vec{w}-\vec{x}^{(-)}\cdot\vec{w})}{\|\vec{w}\|}=\frac{(1-b-(-1-b))}{\|\vec{w}\|}=\frac{2}{\|\vec{w}\|}$$

Distance d

$$max(d)=max\frac{2}{\|\vec{w}\|}=min\|\vec{w}\| \rightarrow \boxed{min\frac{1}{2}\|\vec{w}\|^2}$$

Final objective function

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Optimization: Lagrange primal function

- Using the given data samples (x), find w and b that minimize the objective function with an inequality constraint.

- objective

$$\min \frac{1}{2}\|\vec{w}\|^2$$

- constraint

$$y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)-1\geq 0 \qquad y^{(i)}=\begin{cases} 1 & for \quad "+" \\ -1 & for \quad "-" \end{cases}$$

- inequality constrained optimization problem

$$\min_x f(x), \quad \text{subject to} \quad h(x)\leq 0$$

- Lagrange primal function

$$L_p(x,\lambda)=f(x)+\lambda h(x) \;\rightarrow\; \lambda\geq 0$$

$$\min_x L_p(x,\lambda), \quad s.t \quad h(x)\leq 0$$

$$\frac{\partial L_p}{\partial x} = 0$$

- Lagrange primal function for SVM

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

$$\frac{\partial L_p}{\partial \vec{w}} = 0 \;\rightarrow\; \vec{w}=\sum_{i=1}^{N}\lambda_i y^{(i)}\vec{x}^{(i)}$$

$$\frac{\partial L_p}{\partial b} = 0 \;\rightarrow\; \sum_{i=1}^{N}\lambda_i y^{(i)} = 0$$

$$\lambda_i\geq 0$$

MX-AI

Optimization: Lagrange dual function

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

- Lagrange primal function

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

$$L_D = \frac{1}{2}\left(\sum_{i=1}^{N} \lambda_i y^{(i)} \vec{x}^{(i)}\right)\left(\sum_{j=1}^{N} \lambda_j y^{(j)} \vec{x}^{(j)}\right) + \sum_{i=1}^{N}\left[\lambda_i - \lambda_i y^{(i)}\left(\sum_{i=1}^{N} \lambda_i y^{(i)} \vec{x}^{(i)}\right)\cdot\vec{x}^{(i)} + \lambda_i y^{(i)} b\right]$$

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

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

- Lagrange dual function

- SVM dual problem

$$\underset{\lambda_i}{argmax}\left(-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y^{(i)} y^{(j)} \vec{x}^{(i)}\cdot\vec{x}^{(j)} + \sum_{i=1}^{N} \lambda_i\right)$$

or

$$\underset{\lambda_i}{argmin}\left(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y^{(i)} y^{(j)} \vec{x}^{(i)}\cdot\vec{x}^{(j)} - \sum_{i=1}^{N} \lambda_i\right)$$

subject to: $\lambda_i \geq 0$

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

This is an optimization problem with inequality and equality constraints. As we did in the previous topic, Convex Optimization, we can find the lambdas by solving the QP problem with CVXOPT..

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Optimization: KKT condition and Strong duality

- In general, the primal solution is greater than or equal to the dual solution ($p^* \geq d^*$). If the optimization problem satisfies the KKT conditions, strong duality holds ($p^* = d^*$). For more details, please refer to the convex optimization video, [MXML-5-04].
- The optimization problem for SVM satisfies the KKT conditions as follows. So the solution for dual problem becomes the primal solution.

▪ KKT condition

$$\min_x f(x) \quad \leftarrow \text{Convex function}$$

$$\text{subject to} \quad g_i(x) \leq 0, \quad (i=1,2,\ldots,m)$$

$$h_j(x)=0, \quad (j=1,2,\ldots,n)$$

$$L(x,\lambda,\mu) = f(x)+\sum_{i=1}^{m}\lambda_i g_i(x)+\sum_{j=1}^{n}\mu_i h_i(x), \quad (\lambda_i \geq 0)$$

1) Stationality $\quad \nabla_x L(x^*,\lambda^*,\mu^*) = 0$

2) Complementary slackness $\quad \lambda_i g_i(x^*) = 0$

3) Primal feasibility $\quad g_i(x^*) \leq 0, \quad h_j(x^*)=0$

4) Dual feasibility $\quad \lambda_i \geq 0$

(2)–(4): for all i, j

▪ Lagrange primal function

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

▪ KKT conditions

1) $\dfrac{\partial L_p}{\partial \vec{w}} = 0, \quad \dfrac{\partial L_p}{\partial b} = 0$

2) $\lambda_i\{1-y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)\}=0$ ← If x is outside the positive or negative boundary, λ=0, so it holds true. And if x is on the boundary, λ>0, but the {.} part is 0, so it holds also true.

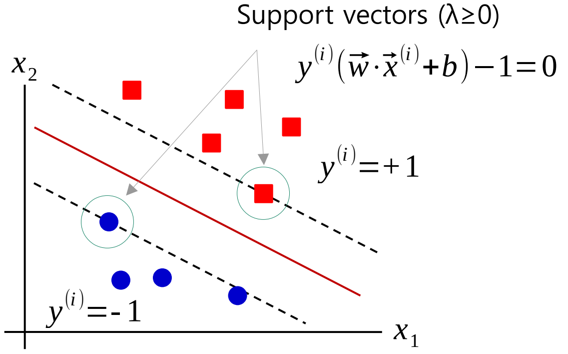
3) $1-y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b) \leq 0$ ← constraints

4) $\lambda \geq 0$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

Decision function

- In the training stage, a decision function is created through the procedure below. And in the prediction stage, this decision function is used to estimate the class of the test data.



1. Solve the QP and find the optimal λ

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

2. Use the optimal λ to find w*.

$$\frac{\partial L_p}{\partial \vec{w}}=0 \rightarrow \vec{w}^*=\sum_{i=1}^{N}\lambda_i^* y^{(i)}\vec{x}^{(i)}$$

3. Use w* to find b*.

3-1. Method-1: Bishop, Pattern Recognition and Machine Learning, p.330, equation (7.18)

$y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)-1=0$ ← Multiply both sides by y(i). $y^{(i)2}=1$.

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

$b^*=y^{(i)}-\vec{w}\cdot\vec{x}^{(i)}$ ← Calculate each b for support vectors and then average them.

3-2. Method-2: Andrew Ng's CS229 Lecture notes-3 eq. (11)

- Calculate b with two support vectors

$$b^*=-\frac{max_{i:y^{(i)}=-1}\vec{w}^*\cdot\vec{x}^{(i)}+min_{i:y^{(i)}=+1}\vec{w}^*\cdot\vec{x}^{(i)}}{2}$$

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

$\vec{w}\cdot\vec{x}^{(i)+}+b=1$ ← "+" sample

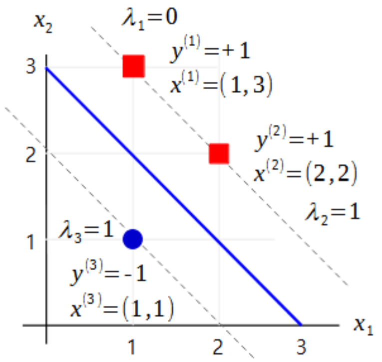
$\vec{w}\cdot\vec{x}^{(i)-}+b=-1$ ← "-" sample

By adding the two equations above, you can find b using the equation on the left.

4. Decision function : $\hat{y}=w_1^*x_1+w_2^*x_2+b^*$

5. Put the test data into the decision function. If y-hat is positive, it is classified as +1, and if it is negative, it is classified as -1.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai



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

$$\text{s.t}\quad \lambda_i\geq 0$$

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

6. Support Vector Machine (SVM)

Part 2: Linear Hard Margin - Implementation

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

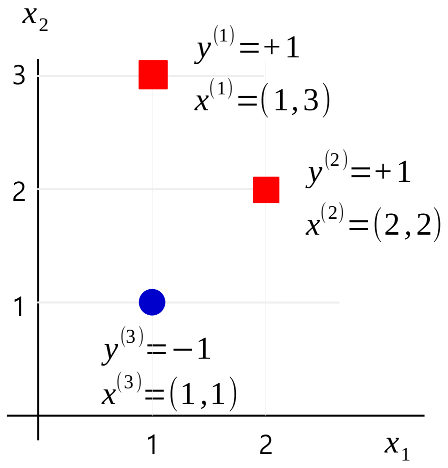
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Example of SVM: Solving by hand

- Given three observed data points, we use SVM to find the decision boundary, or decision function, as shown below.
- Step-1 : Find the λ from Lagrange dual function.

- training data



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

$$L_D=\lambda_1+\lambda_2+\lambda_3-\frac{1}{2}\times$$
$$(\lambda_1\lambda_1 y^{(1)}y^{(1)}\vec{x}^{(1)}\cdot\vec{x}^{(1)}+\lambda_1\lambda_2 y^{(1)}y^{(2)}\vec{x}^{(1)}\cdot\vec{x}^{(2)}+\lambda_1\lambda_3 y^{(1)}y^{(3)}\vec{x}^{(1)}\cdot\vec{x}^{(3)}+$$
$$\lambda_2\lambda_1 y^{(2)}y^{(1)}\vec{x}^{(2)}\cdot\vec{x}^{(1)}+\lambda_2\lambda_2 y^{(2)}y^{(2)}\vec{x}^{(2)}\cdot\vec{x}^{(2)}+\lambda_2\lambda_3 y^{(2)}y^{(3)}\vec{x}^{(2)}\cdot\vec{x}^{(3)}+$$
$$\lambda_3\lambda_1 y^{(3)}y^{(1)}\vec{x}^{(3)}\cdot\vec{x}^{(1)}+\lambda_3\lambda_2 y^{(3)}y^{(2)}\vec{x}^{(3)}\cdot\vec{x}^{(2)}+\lambda_3\lambda_3 y^{(3)}y^{(3)}\vec{x}^{(3)}\cdot\vec{x}^{(3)})$$

$$L_D=\lambda_1+\lambda_2+\lambda_3-\frac{1}{2}(10\lambda_1^2+8\lambda_1\lambda_2-4\lambda_1\lambda_3+$$
$$8\lambda_1\lambda_2+8\lambda_2^2-4\lambda_2\lambda_3-$$
$$4\lambda_1\lambda_3-4\lambda_2\lambda_3+2\lambda_3^2)$$

$$L_D=\lambda_1+\lambda_2+\lambda_3-\frac{1}{2}(10\lambda_1^2+8\lambda_2^2+2\lambda_3^2+$$
$$16\lambda_1\lambda_2-8\lambda_1\lambda_3-8\lambda_2\lambda_3)$$

$$\frac{\partial L_p}{\partial b}=0 \rightarrow \sum_{i=1}^{N}\lambda_i y^{(i)}=\lambda_1+\lambda_2-\lambda_3=0$$

$$L_D=2\lambda_1+2\lambda_2-\frac{1}{2}\times$$
$$(10\lambda_1^2+8\lambda_2^2+2(\lambda_1+\lambda_2)^2+$$
$$16\lambda_1\lambda_2-8\lambda_1(\lambda_1+\lambda_2)-8\lambda_2(\lambda_1+\lambda_2))$$

$$L_D=2\lambda_1+2\lambda_2-2\lambda_1^2-\lambda_2^2-2\lambda_1\lambda_2$$

$$\frac{\partial L_D}{\partial \lambda_1}=2-4\lambda_1-2\lambda_2=0$$

$$\frac{\partial L_D}{\partial \lambda_2}=2-2\lambda_1-2\lambda_2=0$$

$$\lambda_1=0,\quad \lambda_2=1,\quad \lambda_3=1$$

- x with λ > 0 is the support vector.
- Support vector = $x^{(2)}$, $x^{(3)}$

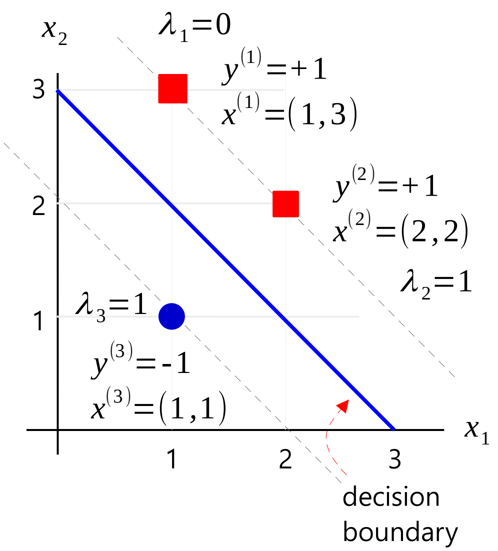
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Example of SVM: Solving by hand

- Step-2 : Find w by substituting λ into the equation obtained by differentiating Lagrange primal function.
- Step-3 : Find b using the support vectors (λ > 0).

- training data



- Find w using λ

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

$$\vec{w}=0\times1\times[1,3]+1\times1\times[2,2]+1\times(-1)\times[1,1]$$

$$\vec{w}=[1,1]$$

- Find b using w and support vectors

$$y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)-1=0$$ ← The support vectors are on this straight line.

$$1\times([1,1]\cdot[2,2]+b)-1=0 \;\rightarrow\; b=-3$$

$$(-1)\times([1,1]\cdot[1,1]+b)-1=0 \;\rightarrow\; b=-3$$

- Decision boundary : $x_1 + x_2-3=0$
- Margin : $d=\frac{2}{\|\vec{w}\|}=\frac{2}{\sqrt{1^2+1^2}}=\sqrt{2}$
- positive boundary: $y(\vec{w}\cdot\vec{x}+b)-1=0,\ (y=+1)$

$$1([1,1]\cdot[x_1,x_2]-3)-1 = x_1+x_2-4=0$$

- negative boundary: $y(\vec{w}\cdot\vec{x}+b)-1=0,\ (y=-1)$

$$(-1)([1,1]\cdot[x_1,x_2]-3)-1 = x_1+x_2-2=0$$

- Classify the test data

test data $\hat{y}=x_1 + x_2-3$

x_1	x_2	$\hat{y}$	predicted class	Remark
3	2	2	+1	2 > 0, classify it as +1
1	0.5	-1.5	-1	-1.5 < 0, classify it as -1
2	0	-1	-1	on the negative boundary
3	1	1	+1	on the positive boundary
3	0	0	?	on the decision boundary

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Quadratic Programming (QP) : using CVXOPT

- Convert the Lagrange dual function to the standard form of Quadratic Programming.
- Reference: https://xavierbourretsicotte.github.io/SVM_implementation.html

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

$$H_{i,j}=y^{(i)}y^{(j)}\vec{x}^{(i)}\cdot\vec{x}^{(j)}$$ ← definition

$$H=\begin{bmatrix} y^{(1)}y^{(1)} & y^{(1)}y^{(2)} \\ y^{(2)}y^{(1)} & y^{(2)}y^{(2)} \end{bmatrix}\times\begin{bmatrix} \vec{x}^{(1)}\cdot\vec{x}^{(1)} & \vec{x}^{(1)}\cdot\vec{x}^{(2)} \\ \vec{x}^{(2)}\cdot\vec{x}^{(1)} & \vec{x}^{(2)}\cdot\vec{x}^{(2)} \end{bmatrix}$$ ← For N=2 element wise product

```
H = np.outer(y, y) * np.dot(x, x.T)
```
← Python code

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

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

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

$$\text{s.t}\quad \lambda_i\geq 0$$

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

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Quadratic Programming (QP) : using CVXOPT

▪ Standard form of QP

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

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

▪ Lagrange dual function for SVM

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

$$\text{s.t}\quad -\lambda_i \le 0,\quad \sum_{i=1}^{N} \lambda_i y^{(i)} = 0 \longleftarrow$$

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

$G := -I$ size = N x N

$h := \vec{0}$ size = N x 1

$A := y$ size = N x 1

$b := 0$ scalar

$$G \cdot \lambda \le h$$

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

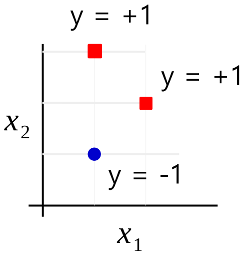
```
# Calculate H matrix
H = np.outer(y, y) * np.dot(x, x.T)

# Construct the matrices required for QP
# in standard form.
n = x.shape[0]
P = cvxopt_matrix(H)
q = cvxopt_matrix(-np.ones((n, 1)))
G = cvxopt_matrix(-np.eye(n))
h = cvxopt_matrix(np.zeros(n))
A = cvxopt_matrix(y.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implementation of SVM using CVXOPT

```
# [MXML-6-02] 1.cvxopt(hard_margin).py
from cvxopt import matrix as matrix
from cvxopt import solvers as solvers
import numpy as np

# 3 data points.
x = np.array([[1., 3.], [2., 2.], [1., 1.]])
y = np.array([[1.], [1.], [-1.]])

# Calculate H matrix
H = np.outer(y, y) * np.dot(x, x.T)

# Construct the matrices required for QP in standard form.
n = x.shape[0]
P = matrix(H)
q = matrix(-np.ones((n, 1)))
G = matrix(-np.eye(n))
h = matrix(np.zeros(n))
A = matrix(y.reshape(1, -1))
b = matrix(np.zeros(1))

# solver parameters
solvers.options['abstol'] = 1e-10
solvers.options['reltol'] = 1e-10
solvers.options['feastol'] = 1e-10

# Perform QP
sol = solvers.qp(P, q, G, h, A, b)

# the solution of the QP, λ
lamb = np.array(sol['x'])
```



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

```
# Calculate w using the lambda, which is the solution to QP.
w = np.sum(lamb * y * x, axis=0).reshape(1, -1)

# Find support vectors
sv_idx = np.where(lamb > 1e-5)[0]
sv_lamb = lamb[sv_idx]
sv_x = x[sv_idx]
sv_y = y[sv_idx].reshape(1, -1)

# Calculate b using the support vectors and calculate the average.
# Reference: Bishop, Pattern Recognition and Machine Learning, p.330,
# equation (7.18)
b = sv_y - np.dot(w, sv_x.T)
b = np.mean(b)

print('\nlambda =', np.round(lamb.flatten(), 3))
print('w =', np.round(w, 3))
print('b =', np.round(b, 3))
```

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

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

```
     pcost       dcost       gap    pres   dres
 0: -7.6444e-01 -1.9378e+00  1e+00  1e-16  2e+00
 1: -9.1982e-01 -1.0024e+00  8e-02  1e-16  3e-01
 ...
 6: -1.0000e+00 -1.0000e+00  3e-06  2e-16  3e-16
 7: -1.0000e+00 -1.0000e+00  4e-07  2e-16  3e-16
 Optimal solution found.

Lambda = [ 0.  1.  1.]
     w = [ 1.  1.]
     b = -3
```

It matches well with the results solved by hand.

Decision boundary: $x_1 + x_2 - 3 = 0$

MX-AI

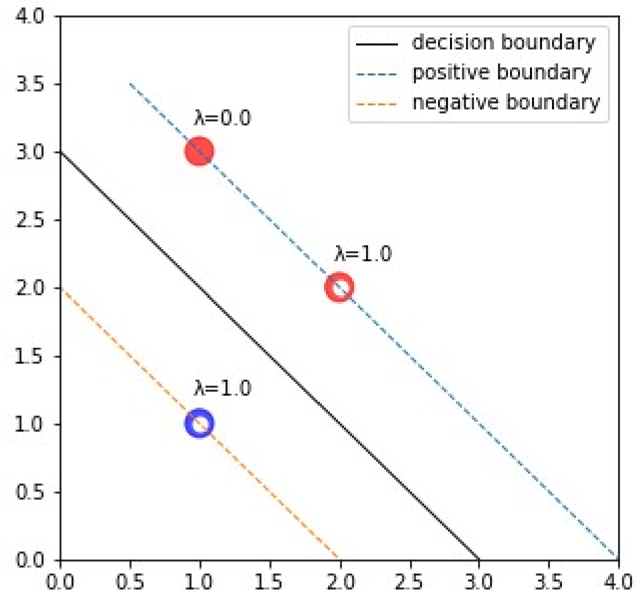
Implementation of SVM using CVXOPT

```
# Visualize the data points
plt.figure(figsize=(5,5))
color= ['red' if a == 1 else 'blue' for a in y]
plt.scatter(x[:, 0], x[:, 1], s=200, c=color, alpha=0.7)
plt.xlim(0, 4)
plt.ylim(0, 4)

# Visualize the decision boundary
x1_dec = np.linspace(0, 4, 50).reshape(-1, 1)
x2_dec = -(w[0][0] / w[0][1]) * x1_dec - b / w[0][1]
plt.plot(x1_dec, x2_dec, c='black', lw=1.0, label='decision boundary')

# Visualize the positive & negative boundary
w_norm = np.sqrt(np.sum(w ** 2))
w_unit = w / w_norm
half_margin = 1 / w_norm
upper = np.hstack([x1_dec, x2_dec]) + half_margin * w_unit
lower = np.hstack([x1_dec, x2_dec]) - half_margin * w_unit

plt.plot(upper[:, 0], upper[:, 1], '--', lw=1.0, label='positive boundary')
plt.plot(lower[:, 0], lower[:, 1], '--', lw=1.0, label='negative boundary')

# Visualize the support vectors
plt.scatter(sv_x[:, 0], sv_x[:, 1], s=50, marker='o', c='white')
for s, (x1, x2) in zip(lamb, x):
    plt.annotate('λ=' + str(s[0].round(2)), (x1-0.05, x2 + 0.2))

plt.legend()
plt.show()
```

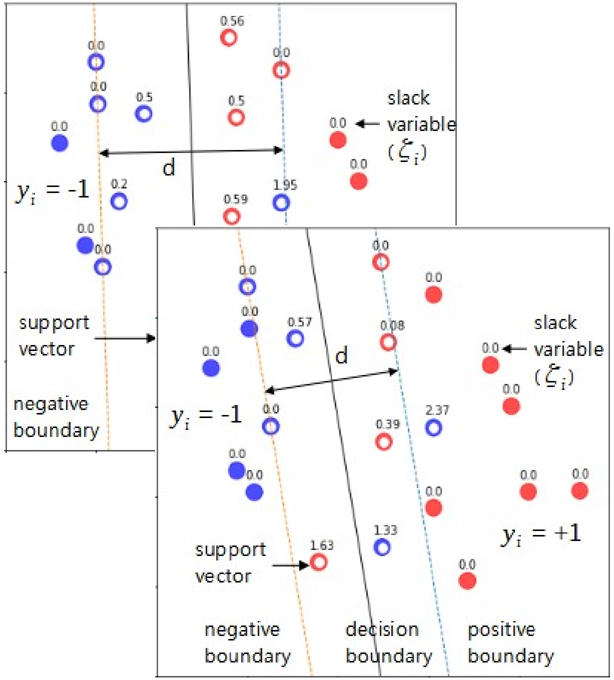
$$\|\vec{w}\|=\sqrt{w_1^2+w_2^2}$$

$\frac{\vec{w}}{\|\vec{w}\|}$: the unit normal vector $\quad \frac{d}{2}=\frac{1}{\|\vec{w}\|}$

```
print("\nMargin = {:.4f}".\
      format(half_margin * 2))

Margin = 1.4142
```



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai



6. Support Vector Machine (SVM)

Part 3: Linear Soft Margin - Algorithm

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

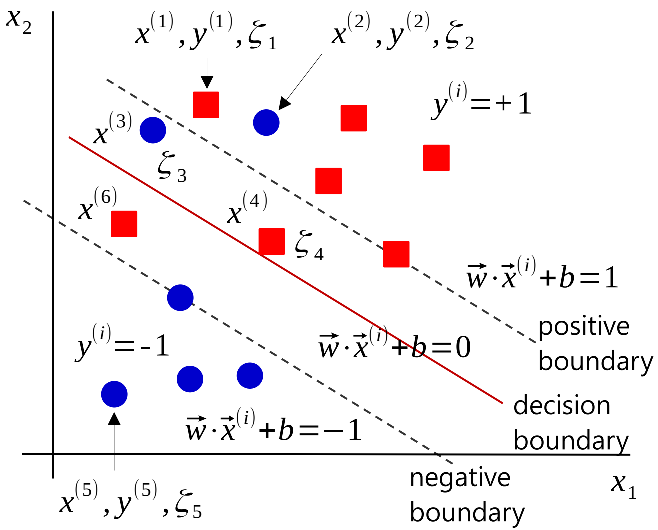
www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Linear Soft Margin – Slack variable

- If the classes +1 and -1 are mixed and cannot be completely separated linearly, they can be linearly separated by allowing for some errors. It is called linear soft margin SVM.
- We assign a slack variable (ξ) to each data point to allow for misclassification.
- Our goal is to find a decision boundary that minimizes the total error = ∑ξ,, (ξ ≥ 0) while maximizing the margin.



$\vec{w}\cdot\vec{x}^{(1)}+b>1$	$0<\vec{w}\cdot\vec{x}^{(3)}+b<1$	$\vec{w}\cdot\vec{x}^{(5)}+b<-1$
$\vec{w}\cdot\vec{x}^{(2)}+b>1$	$0<\vec{w}\cdot\vec{x}^{(4)}+b<1$	$-1<\vec{w}\cdot\vec{x}^{(6)}+b<0$

- Constraints

$$\begin{cases} \vec{w}\cdot\vec{x}^{(i)}+b\geq 1-\zeta_i & if\ y^{(i)}=+1 \\ \vec{w}\cdot\vec{x}^{(i)}+b\leq -1+\zeta_i & if\ y^{(i)}=-1 \end{cases}$$

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

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

For correctly classified data, x(1), x(4), x(5), ξ is greater than or equal to 0. For misclassified data, x(2), x(3), x(6), ξ is greater than or equal to 1. The data point x(4) that is classified correctly but falls between the positive and negative boundaries will have a ξ value between 0 and 1. Therefore, ξ can be considered a measure of misclassification.

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

$$\zeta_1=max(0,1-\underbrace{(\vec{w}\cdot\vec{x}^{(1)}+b)}_{>1}) = 0$$

$$\zeta_2=max(0,1+\underbrace{(\vec{w}\cdot\vec{x}^{(2)}+b)}_{>1}) > 2$$

$$\zeta_3=max(0,1+\underbrace{(\vec{w}\cdot\vec{x}^{(3)}+b)}_{0\sim 1}) = 1\sim 2$$

$$\zeta_4=max(0,1-\underbrace{(\vec{w}\cdot\vec{x}^{(4)}+b)}_{0\sim 1}) = 0\sim 1$$

$$\zeta_5=max(0,1+\underbrace{(\vec{w}\cdot\vec{x}^{(5)}+b)}_{<-1}) = 0$$

$$\zeta_6=max(0,1-\underbrace{(\vec{w}\cdot\vec{x}^{(6)}+b)}_{-1\sim 0}) = 1\sim 2$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Objective function

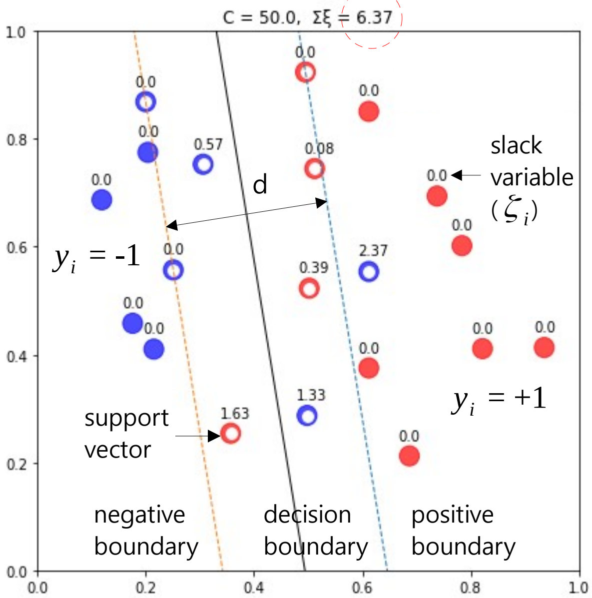
- Soft margin is a method that allows for some errors. Two objective functions must be considered together. The first is to maximize margin, and the second is to minimize errors. The two goals are trade-offs and must be properly balanced.
- The two objective functions are combined into one by the weight, C.

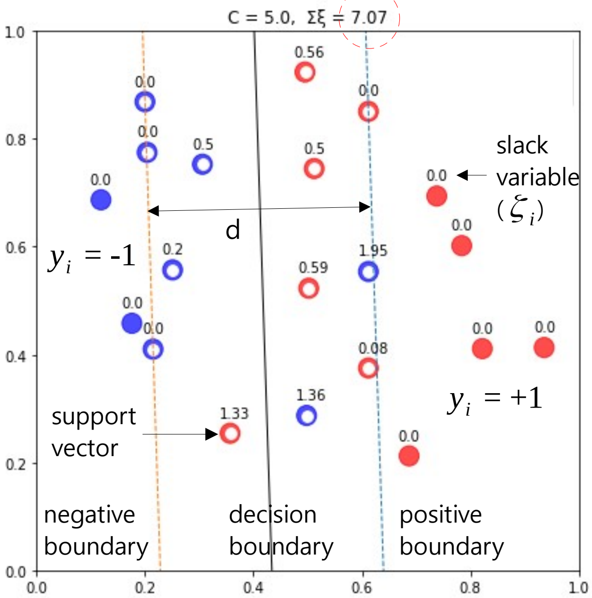
1) Goal-1: $max(d)=min\frac{1}{2}||\vec{w}||^2$

2) Goal-2: $min\sum_{i=1}^{N}\zeta_i=min\sum_{i=1}^{N}max(0,1-y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b))$ → hinge loss

Objective function: $min(\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\zeta_i^k)$

Because C is large, the overall error is smaller than the error on the right.





- If C is small, the margin is made large even if the error is large. If C is large, the error is made small even if the margin is small.
- Typically k=1, 2 is used. (k = 1 : hinge loss, k = 2 : squared hinge loss).
- The first term can be viewed as a loss and the second term as a regularized term (penalty term).
- Conversely, the second term can be viewed as a loss (hinge loss) and the first term as a regularized term (L2, Ridge).

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Hinge loss

- Slack variables, ξ, can be considered errors.
- In regression, the further a data point is from the regression curve, above or below, the larger the error. However, in classification, the error increases the further a data point is from the boundary in the wrong direction, but the error is zero no matter how far the data point is in the right direction. This is called hinge loss.
- The typical form of hinge loss is something like max(0, something).

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

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

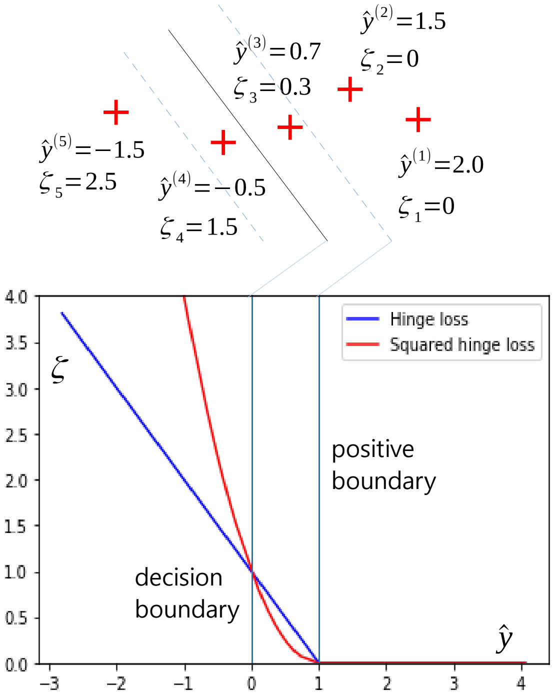
typical error

$$y^{(i)}=+1 \rightarrow \xi_i=max(0,y^{(i)}-\hat{y}^{(i)})=max(0,actual-predict)$$

$$y^{(i)}=\text{-}1 \rightarrow \xi_i=max(0,\hat{y}^{(i)}-y^{(i)})=max(0,predict-actual)$$

Correctly classified data has all errors of zero.

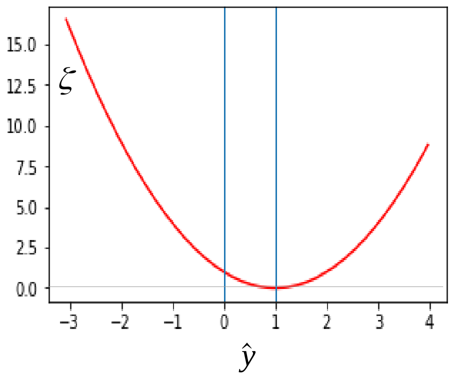
- Hinge & squared hinge loss for (+) sample



- Quadratic loss

$$y^{(i)}=\pm 1 \rightarrow \xi_i^2=(\hat{y}^{(i)}-y^{(i)})^2$$

The squared (quadratic) loss shows the following characteristics. The farther left or right you go from the boundary, the larger the error becomes.



Objective function

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i]$$

L2-regularized (or penalty) & hinge loss

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i^2]$$

L2-regularized & squared hinge loss
It is more sensitive to errors far from the boundary.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Optimization: Lagrange primal function

- objective

$$\min\left[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\zeta_i\right]$$

- constraints

$$1-\zeta_i-y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)\le 0 \qquad y^{(i)}=\begin{cases}1 & \text{for "+"}\\ -1 & \text{for "-"}\end{cases} \qquad -\zeta\le 0$$

- inequality constrained optimization problem

$$\min_x f(x), \quad s.t \quad h(x)\le 0$$

- Lagrangian primal function

$$L_p(x,\lambda)=f(x)+\lambda h(x) \ \rightarrow\ \lambda\ge 0$$

$$\min_x L_p(x,\lambda), \quad s.t \quad h(x)\le 0$$

$$\frac{\partial L_p}{\partial x} = 0$$

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{N}\zeta_i+\sum_{i=1}^{N}\lambda_i\{1-\zeta-y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)\}+\sum_{i=1}^{N}\{-\mu_i\zeta_i\} \qquad (\zeta\ge 0,\ \lambda_i\ge 0,\ \mu_i\ge 0)$$

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

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

$$\frac{\partial L_p}{\partial \zeta_i} = C-\lambda_i-\mu_i=0 \ \rightarrow C=\lambda_i+\mu_i \qquad \lambda_i=C-\mu_i \ \rightarrow\ 0\le\lambda_i\le C$$

MX-AI

Optimization: Lagrange dual function

- The Lagrange dual function for the soft margin is obtained in the same way as for the hard margin.
- The dual function is the same as that of the hard margin, but has different constraints.

Primal function

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{N}\zeta_i-\sum_{i=1}^{N}\lambda_i\{y^{(i)}(\vec{w}\cdot\vec{x}^{(i)}+b)-1+\zeta\}-\sum_{i=1}^{N}\mu_i\zeta_i \qquad (\zeta\geq 0,\ \lambda_i\geq 0,\ \mu_i\geq 0)$$

$$\vec{w}=\sum_{i=1}^{N}\lambda_i y^{(i)}\vec{x}^{(i)} \qquad \sum_{i=1}^{N}\lambda_i y^{(i)}=0 \qquad C-\lambda_i-\mu_i=0 \rightarrow C=\lambda_i+\mu_i \rightarrow 0\leq\lambda_i\leq C$$

$$L_D=\frac{1}{2}\left(\sum_{i=1}^{N}\lambda_i y^{(i)}\vec{x}^{(i)}\right)\left(\sum_{j=1}^{N}\lambda_j y^{(j)}\vec{x}^{(j)}\right)+C\sum_{i=1}^{N}\zeta_i-\sum_{i=1}^{N}\left[\lambda_i y^{(i)}\left(\sum_{i=1}^{N}\lambda_i y^{(i)}\vec{x}^{(i)}\right)\cdot\vec{x}^{(i)}+\lambda_i y^{(i)}b-\lambda_i+\lambda_i\zeta_i\right]-\sum_{i=1}^{N}\mu_i\zeta_i$$

$$L_D=\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y^{(i)}y^{(j)}\vec{x}^{(i)}\cdot\vec{x}^{(j)}+\cancel{\sum_{i=1}^{N}(C-\lambda_i-\mu_i)\zeta_i}+\sum_{i=1}^{N}\lambda_i-\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y^{(i)}y^{(j)}\vec{x}^{(i)}\cdot\vec{x}^{(j)}+\cancel{b\sum_{i=1}^{N}\lambda_i y^{(i)}}$$

$$\boxed{\begin{aligned}&L_D=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y^{(i)}y^{(j)}\vec{x}^{(i)}\cdot\vec{x}^{(j)}\\ &\text{constraints:}\quad 0\leq\lambda_i\leq C,\ \sum_{i=1}^{N}\lambda_i y^{(i)}=0\end{aligned}}$$

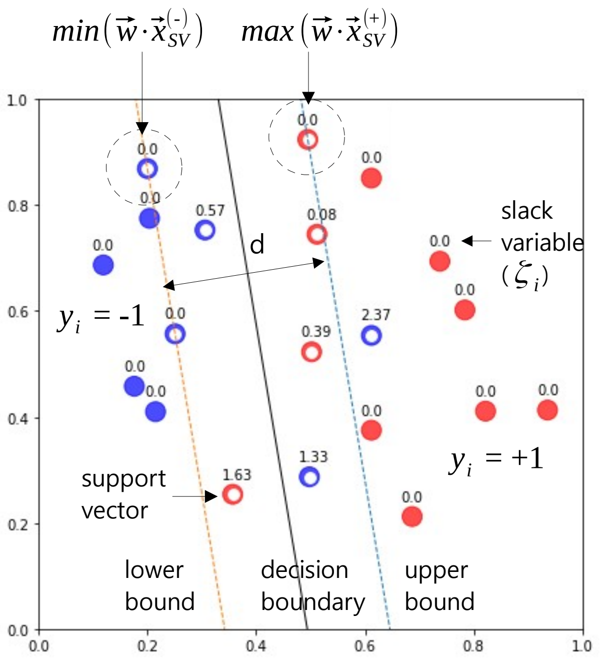
- These are the Lagrange dual function and the constraints for the linear soft margin SVM.
- As in the hard margin SVM, the λ that maximizes this equation can be obtained through quadratic programming.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Decision function

- We can find the lambda from the dual function, and then use the lambda to find w. And we can use the support vectors to find b.
- We use the w and b to obtain the decision function. And we use the decision function to predict the class of the test data.



- Use λ to find w

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

- Use the support vectors (SV) to find b. SVs lie between the positive and negative boundaries.

$$\vec{w}\cdot\vec{x}_i^{(+)}+b\geq 1-\xi_i$$ ← "+" sample i satisfies this inequality.

$$\vec{w}\cdot\vec{x}_j^{(-)}+b\leq -1+\xi_j$$ ← "-" sample i satisfies this inequality.

The SV(+) sample with the largest wx is on the positive boundary, and the SV(-) sample with the smallest wx is on the negative boundary (ξi = ξj = 0). Calculate b using these two samples.

$$b=-\frac{max(\vec{w}\cdot\vec{x}_{SV}^{(+)})+min(\vec{w}\cdot\vec{x}_{SV}^{(-)})}{2}$$

- Decision function: We use this function to predict the class of the test data.

$$\hat{y} = w_1x_1 + w_2x_2 + \ldots + b$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

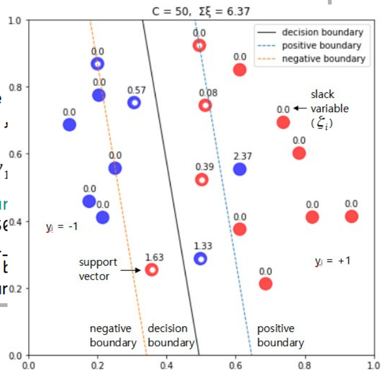
```
# [MXML-6-04] 4.linearSVC(soft_margin).py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC

x = np.array([[0.2,   0.869], [0.687, 0.212],    ...
y = np.array([-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,1, ...
```

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i]$$

```
model = LinearSVC(penalty='l2', loss='hinge', C=50)
# model = LinearSVC(penalty='l2', loss='squared_hinge', C=50)
model.fit(x, y)
```

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i^2]$$

```
# parameters
w = model.coef_[0]
b = model.intercept_[0]

# Visualize the data points
plt.figure(figsize=(7,7))
color= ['red' if a == 1 else
plt.scatter(x[:, 0], x[:, 1],
plt.xlim(0, 1)
plt.ylim(0, 1)

# Visualize the decision bour
x1_dec = np.linspace(0, 1, 5(
x2_dec = -(w[0] / w[1]) * x1_
plt.plot(x1_dec, x2_dec, c='k
         label='decision bour
```



6. Support Vector Machine (SVM)

Part 4: Linear Soft Margin - Implementation

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Quadratic Programming for linear soft margin SVM

- It is the same as for hard margin SVM. However, only the constraints on λ are different.

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

$$H_{i,j}=y^{(i)}y^{(j)}\vec{x}^{(i)}\cdot\vec{x}^{(j)}$$ ← definition

$$H=\begin{bmatrix} y^{(1)}y^{(1)} & y^{(1)}y^{(2)} \\ y^{(2)}y^{(1)} & y^{(2)}y^{(2)} \end{bmatrix}\times\begin{bmatrix} \vec{x}^{(1)}\cdot\vec{x}^{(1)} & \vec{x}^{(1)}\cdot\vec{x}^{(2)} \\ \vec{x}^{(2)}\cdot\vec{x}^{(1)} & \vec{x}^{(2)}\cdot\vec{x}^{(2)} \end{bmatrix}$$ ← For N=2
element wise product

```
H = np.outer(y, y) * np.dot(x, x.T)
```
← Python code

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

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

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

$$\text{s.t}\quad -\lambda_i\le 0,\quad \lambda_i\le C$$

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

This constraint is added to the hard margin SVM.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Quadratic Programming for linear soft margin SVM

- Standard form of QP

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

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

- Lagrange dual function for linear soft margin SVM

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

$$\text{s.t}\quad -\lambda_i \le 0$$
$$\lambda_i \le C$$
$$\sum_{i=1}^{N} \lambda_i y^{(i)} = 0$$

← 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`

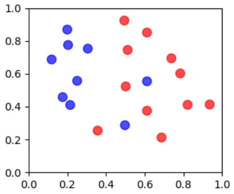
$$G \cdot \lambda \le h \longrightarrow \underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{G} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix}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

Since the constraints have changed, G and h are different from those of the hard margin SVM.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

Implementation of the linear soft margin SVM using CVXOPT

```python
# [MXML-6-04] 2.cvxopt(soft_margin).py
import numpy as np
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
import matplotlib.pyplot as plt

# training data
x = np.array([[0.2,   0.869], [0.687, 0.212],    ...
y = np.array([-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,1, ...
y = y.astype('float').reshape(-1, 1)
C = 50.0
N = x.shape[0]
```



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

```python
# Construct the matrices required for QP in standard form.
H = np.outer(y, y) * np.dot(x, x.T)
P = cvxopt_matrix(H)
q = cvxopt_matrix(np.ones(N) * -1)
A = cvxopt_matrix(y.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))

g = np.vstack([-np.eye(N), np.eye(N)])
G = cvxopt_matrix(g)

h1 = np.hstack([np.zeros(N), np.ones(N) * C])
h = cvxopt_matrix(h1)
```

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ C \\ C \\ C \\ C \end{bmatrix}$$

$$\min\left[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\zeta_i^k\right]$$

```python
# solver parameters
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Perform QP
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
lamb = np.array(sol['x']) # the solution to the QP, λ

# Calculate w using the lambda, which is the solution to QP.
w = np.sum(lamb * y * x, axis=0)

# Find support vectors
sv_idx = np.where(lamb > 1e-5)[0]
sv_lamb = lamb[sv_idx]
sv_x = x[sv_idx]
sv_y = y[sv_idx]

sv_plus = sv_x[np.where(sv_y > 0)[0]]   # '+1' samples
sv_minus = sv_x[np.where(sv_y < 0)[0]]  # '-1' samples

# Calculate b using the support vectors and calculate the average.
b = -(np.max(np.dot(w, sv_plus.T))+np.min(np.dot(w, sv_minus.T))) / 2.0
```

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

$$b=-\frac{\max\left(\vec{w}\cdot\vec{x}_{SV}^{(+)}\right)+\min\left(\vec{w}\cdot\vec{x}_{SV}^{(-)}\right)}{2}$$

■ Implementation of the linear soft margin SVM using CVXOPT

```python
# Visualize the data points
plt.figure(figsize=(7,7))
color= ['red' if a == 1 else 'blue' for a in y]
plt.scatter(x[:, 0], x[:, 1], s=200, c=color, alpha=0.7)
plt.xlim(0, 1)
plt.ylim(0, 1)

# Visualize the decision boundary
x1_dec = np.linspace(0, 1, 50).reshape(-1, 1)
x2_dec = -(w[0] / w[1]) * x1_dec - b / w[1]
plt.plot(x1_dec, x2_dec, c='black', lw=1.0, label='decision boundary')

# display slack variables, slack variable = max(0, 1 - y(wx + b))
y_hat = np.dot(w, x.T) + b
slack = np.maximum(0, 1 - y.flatten() * y_hat)
for s, (x1, x2) in zip(slack, x):
    plt.annotate(str(s.round(2)), (x1-0.02, x2 + 0.03))

# Visualize the positive & negative boundary and support vectors
w_norm = np.sqrt(np.sum(w ** 2))
w_unit = w / w_norm
half_margin = 1 / w_norm
upper = np.hstack([x1_dec, x2_dec]) + half_margin * w_unit
lower = np.hstack([x1_dec, x2_dec]) - half_margin * w_unit
plt.plot(upper[:, 0], upper[:, 1], '--', lw=1.0, label='positive boundary')
plt.plot(lower[:, 0], lower[:, 1], '--', lw=1.0, label='negative boundary')
plt.scatter(sv_x[:, 0], sv_x[:, 1], s=60, marker='o', c='white')
plt.legend()
plt.title('C = ' + str(C) + ',  Σξ = ' + str(np.sum(slack).round(2)))
plt.show()
```

$$w_1 x_1 + w_2 x_2 + b = 0$$

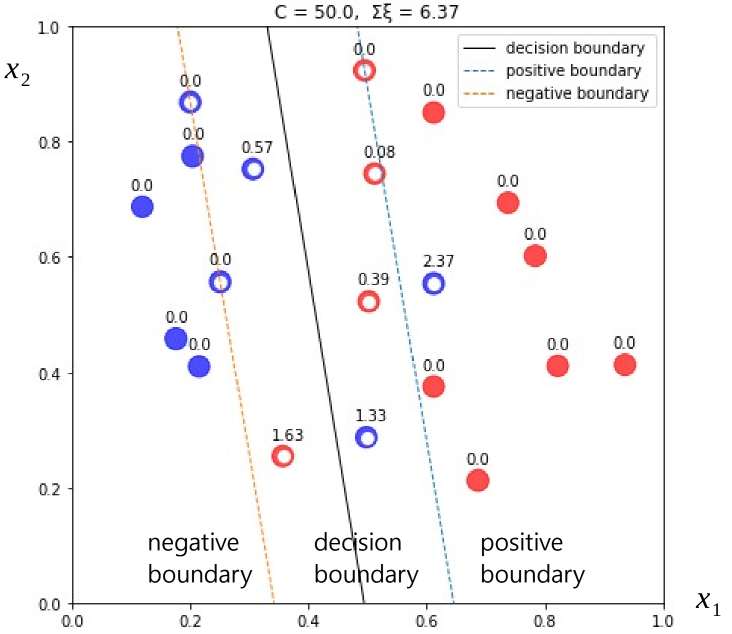
$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

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

$$\frac{d}{2} = \frac{1}{\|\vec{w}\|} \qquad \|\vec{w}\| = \sqrt{w_1^2 + w_2^2} \qquad \frac{\vec{w}}{\|\vec{w}\|}$$

: the unit normal vector

```
     pcost       dcost       gap    pres   dres
 0:  1.2279e+03 -1.3111e+04  1e+04  2e-14  2e-14
 1:  1.1798e+02 -1.5089e+03  2e+03  1e-14  1e-14
 2: -2.2267e+02 -4.9527e+02  3e+02  2e-14  4e-15
 ...
 8: -3.4090e+02 -3.4090e+02  2e-04  1e-14  9e-15
 9: -3.4090e+02 -3.4090e+02  2e-06  2e-14  1e-14
10: -3.4090e+02 -3.4090e+02  2e-08  3e-15  1e-14
Optimal solution found.
```

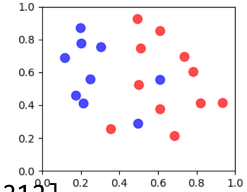


Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implementation of the linear soft margin SVM using scikit-learn's SVC

```
# [MXML-6-04] 3.SVC(soft_margin).py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

x = np.array([[0.2,   0.869], [0.687, 0.212],   ...
y = np.array([-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,1,  ...
C = 50

# Create SVC model and fit it the the training data
model = SVC(C=C, kernel='linear')
model.fit(x, y)

# parameters
w = model.coef_[0]
b = model.intercept_[0]

# Visualize the data points
plt.figure(figsize=(7,7))
color= ['red' if a == 1 else 'blue' for a in y]
plt.scatter(x[:, 0], x[:, 1], s=200, c=color, alpha=0.7)
plt.xlim(0, 1)
plt.ylim(0, 1)

# Visualize the decision boundary
x_dec = np.linspace(0, 1, 50).reshape(-1, 1)
y_dec = -(w[0] / w[1]) * x_dec - b / w[1]
plt.plot(x_dec, y_dec, c='black', lw=1.0,
         label='decision boundary')
```



$$\hat{y}=\vec{w}\cdot\vec{x}+b$$

$$\zeta=max(0,1-y\cdot\hat{y})$$

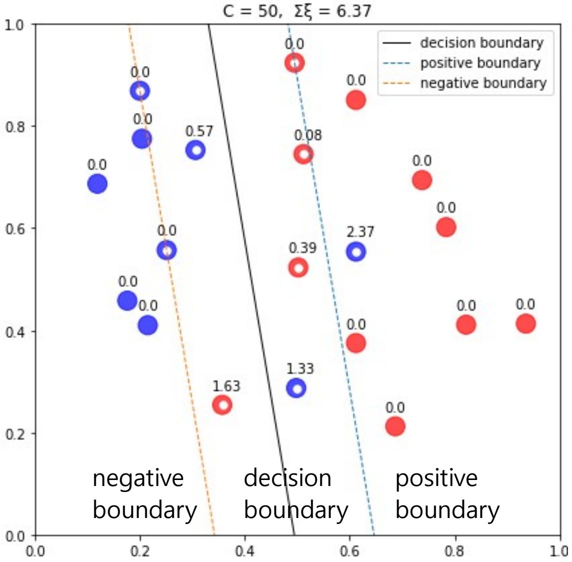
$$w_0x+w_1y+b=0 \qquad y=-\frac{w_0}{w_1}x-\frac{b}{w_1}$$

```
# Visualize the positive & negative boundary
w_norm = np.sqrt(np.sum(w ** 2))
w_unit = w / w_norm
half_margin = 1 / w_norm

upper = np.hstack([x_dec, y_dec]) + half_margin * w_unit
lower = np.hstack([x_dec, y_dec]) - half_margin * w_unit
plt.plot(upper[:, 0], upper[:, 1], '--', lw=1.0, label='positive boundary')
plt.plot(lower[:, 0], lower[:, 1], '--', lw=1.0, label='negative boundary')

# display slack variables, slack variable = max(0, 1 - y(wx + b))
y_hat = np.dot(w, x.T) + b

slack = np.maximum(0, 1 - y * y_hat)
for s, (x1, x2) in zip(slack, x):
    plt.annotate(str(s.round(2)),
    (x1-0.02, x2 + 0.03))

# Visualize support vectors.
sv = model.support_vectors_
plt.scatter(sv[:, 0], sv[:, 1], s=30,
            c='white')
plt.title('C = ' + str(C) + ',
          Σξ = ' +
          str(np.sum(slack).round(2)))
plt.legend()
plt.show()
```

$$\|\vec{w}\|=\sqrt{w_0^2+w_1^2}$$

$$\frac{\vec{w}}{\|\vec{w}\|}$$: unit normal vector

$$\frac{d}{2}=\frac{1}{\|\vec{w}\|}$$



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

- Implementation of the linear soft margin SVM using scikit-learn's LinearSVC

```
# [MXML-6-04] 4.linearSVC(soft_margin).py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC

x = np.array([[0.2,   0.869], [0.687, 0.212],    ...
y = np.array([-1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,1, ...
```

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\zeta_i]$$

```
model = LinearSVC(penalty='l2', loss='hinge', C=50)
# model = LinearSVC(penalty='l2', loss='squared_hinge', C=50)
model.fit(x, y)
```

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\zeta_i^2]$$

```
# parameters
w = model.coef_[0]
b = model.intercept_[0]

# Visualize the data points
plt.figure(figsize=(7,7))
color= ['red' if a == 1 else 'blue' for a in y]
plt.scatter(x[:, 0], x[:, 1], s=200, c=color, alpha=0.7)
plt.xlim(0, 1)
plt.ylim(0, 1)
```

$$w_1x_1+w_2x_2+b=0$$

$$x_2=-\frac{w_1}{w_2}x_1-\frac{b}{w_2}$$

```
# Visualize the decision boundary
x1_dec = np.linspace(0, 1, 50).reshape(-1, 1)
x2_dec = -(w[0] / w[1]) * x1_dec - b / w[1]
plt.plot(x1_dec, x2_dec, c='black', lw=1.0,
         label='decision boundary')
```

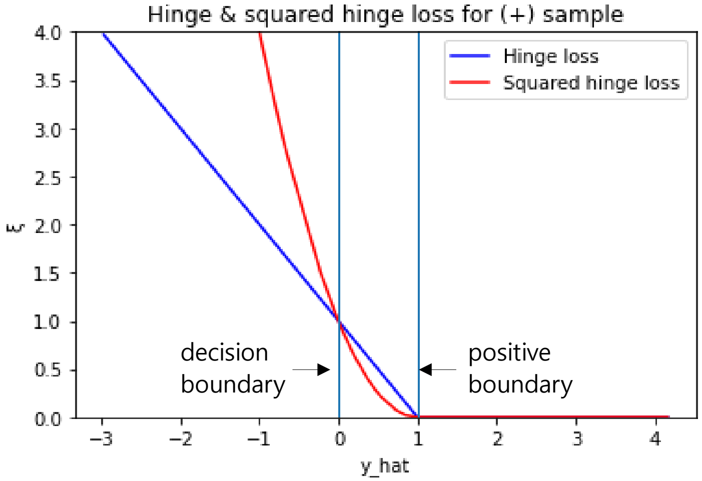
```
# Visualize the positive & negative boundary
w_norm = np.sqrt(np.sum(w ** 2))
w_unit = w / w_norm
half_margin = 1 / w_norm
upper = np.hstack([x1_dec, x2_dec]) + half_margin * w_unit
lower = np.hstack([x1_dec, x2_dec]) - half_margin * w_unit

plt.plot(upper[:, 0], upper[:, 1], '--', lw=1.0,label='positive boundary')
plt.plot(lower[:, 0], lower[:, 1], '--', lw=1.0,label='negative boundary')

# display slack variables, slack variable = max(0, 1 - y(wx + b))
y_hat = np.dot(w, x.T) + b
slack = np.maximum(0, 1 - y * y_hat)
for s, (x1, x2) in zip(slack, x):
    plt.annotate(str(s.round(2)), (x1-0.02, x2 + 0.03))

# Visualize support vectors.
sv = x[np.where(np.abs(y_hat) <= 1.0)[0]]
plt.scatter(sv[:, 0], sv[:, 1], s=30, c='white')

plt.title('C = ' + str(C) + ',  Σξ = ' + str(np.sum(slack).round(2)))
plt.legend()
plt.show()

# Hinge & squared hinge loss plot for [+] samples (y = +1)
x_rand = np.random.rand(100, 2)
y_rand = np.dot(w, x_rand.T) + b      # y_hat for x_rand
s_rand = np.maximum(0, 1 - y_rand)    # slack variables for y_rand
sort_idx = np.argsort(y_rand)
```

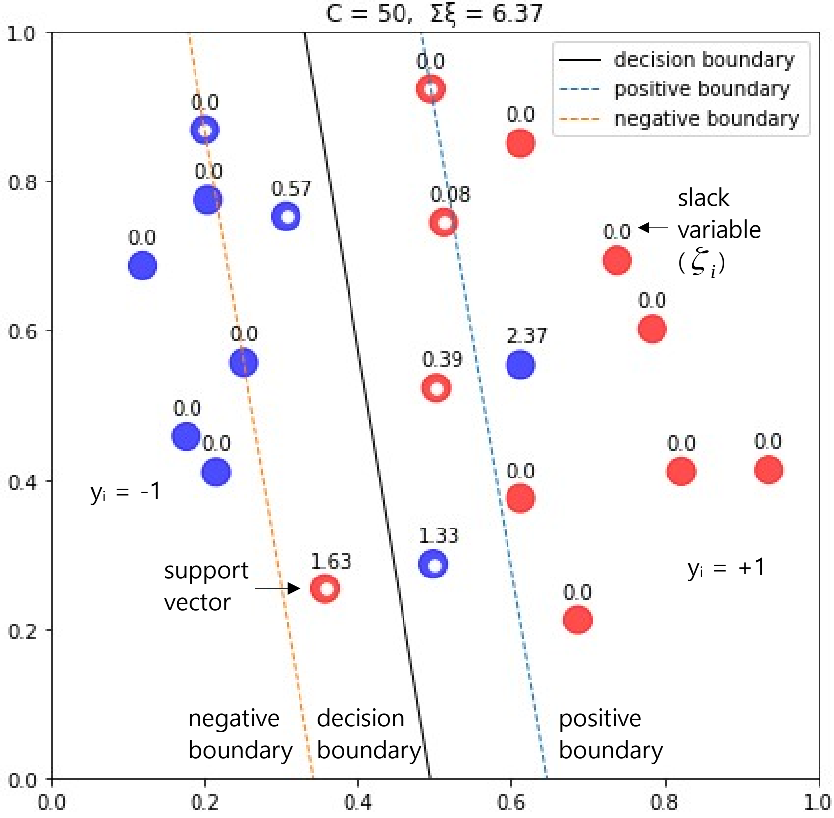
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

- Implementation of the linear soft margin SVM using scikit-learn's LinearSVC

```
y_rand = y_rand[sort_idx]
s_rand = s_rand[sort_idx]

plt.plot(y_rand, s_rand, c='blue', label='Hinge loss')
plt.plot(y_rand, s_rand ** 2, c='red', label='Squared hinge loss')
plt.legend()
plt.axvline(x=0, lw=1)
plt.axvline(x=1, lw=1)
plt.xlabel('y_hat')
plt.ylabel('ξ')
plt.ylim(0, 4)
plt.title('Hinge & squared hinge loss for (+) sample')
plt.show()
```





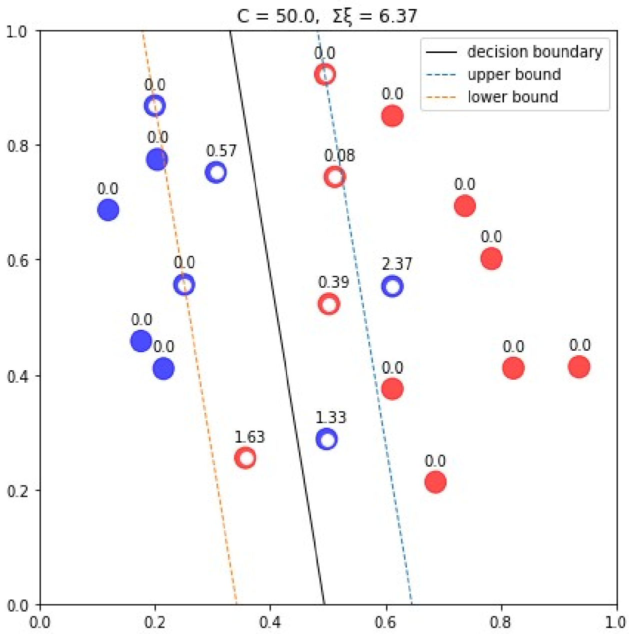
MX-AI

Compare the results

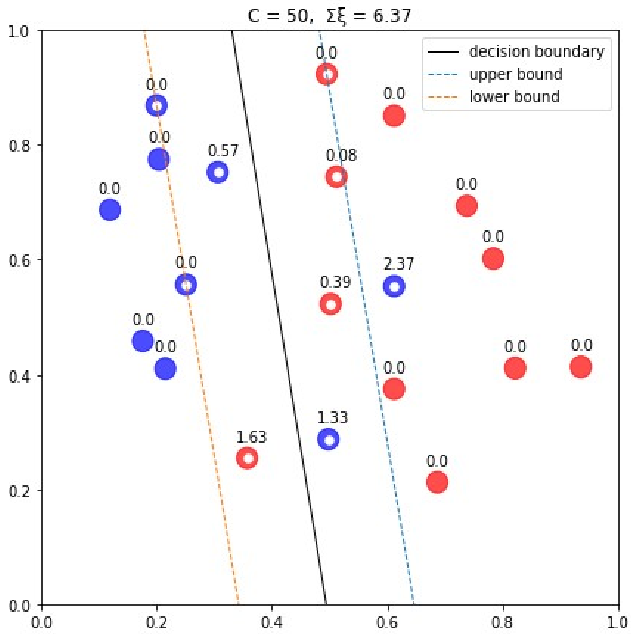
- All three results are the same.

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\zeta_i]$$

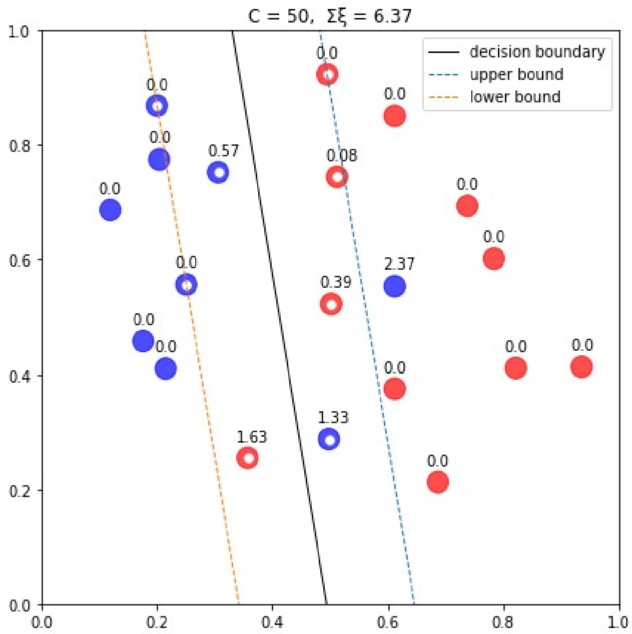
- Result of CVXOPT



- Result of SVC(kernel='linear', C=50)



- LinearSVC(penalty='l2', loss='hinge', C=50)



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

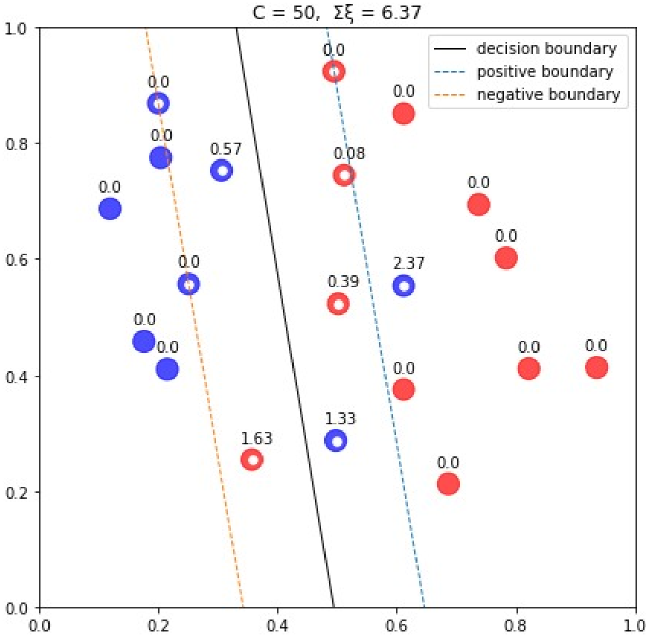
MX-AI

▪ Observation of changes in decision boundary according to changes in C

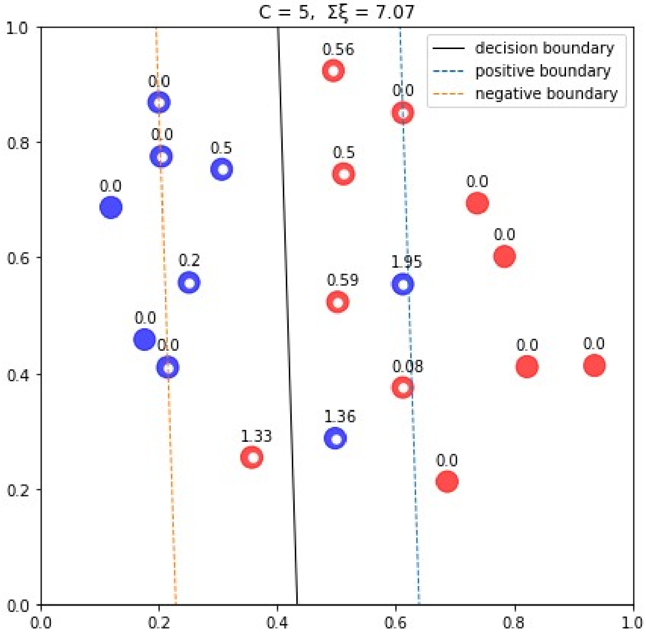
- As C gets smaller, the margin and ∑ξ get bigger. This is because it focuses more on the goal of maximizing the margin. Conversely, as C increases, it focuses more on the goal of minimizing ∑ξ, so ∑ξ decreases but the margin also becomes smaller.

$$min[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i]$$

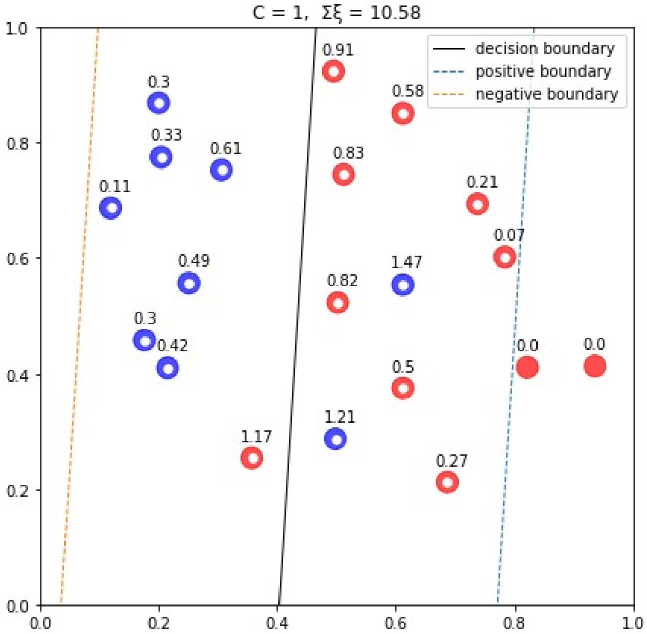
- C = 50, ∑ξ = 6.37



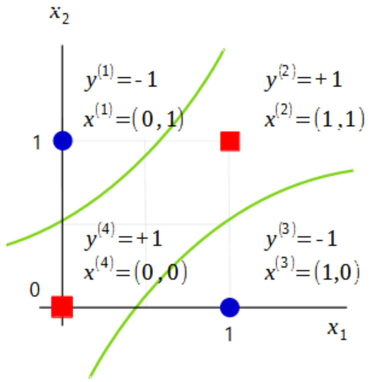
- C = 5, ∑ξ = 7.07

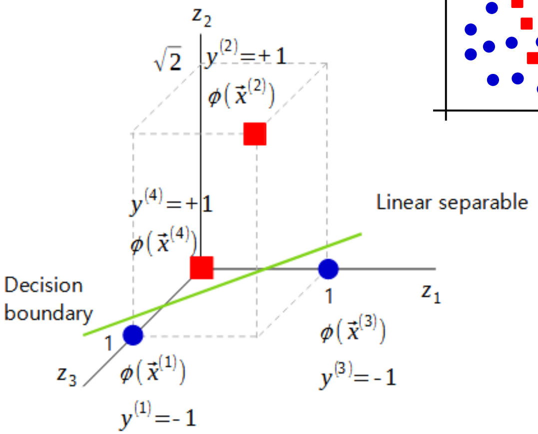


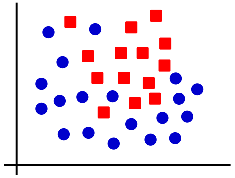
- C = 1, ∑ξ = 10.58



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai







6. Support Vector Machine (SVM)

Part 5: Non-Linear SVM – Kernel trick

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

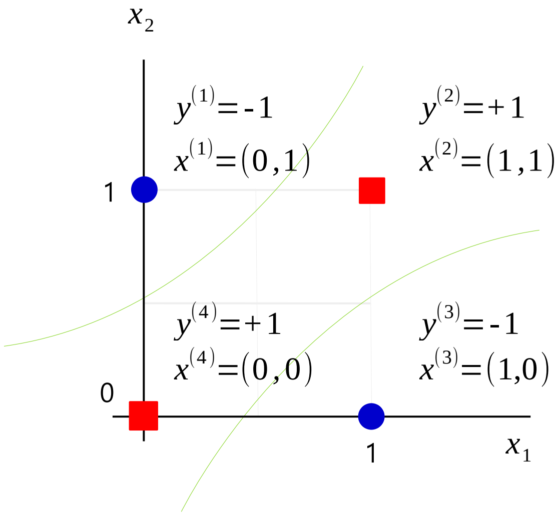
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Non-linear SVM : Feature space transformation

- 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.

Feature space in 2D



Linear separation is not possible.

$$\phi : R^2 \rightarrow R^3$$

$$\vec{x}=(x_1,x_2) \xrightarrow{\phi} \phi(\vec{x}) = (x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2)$$

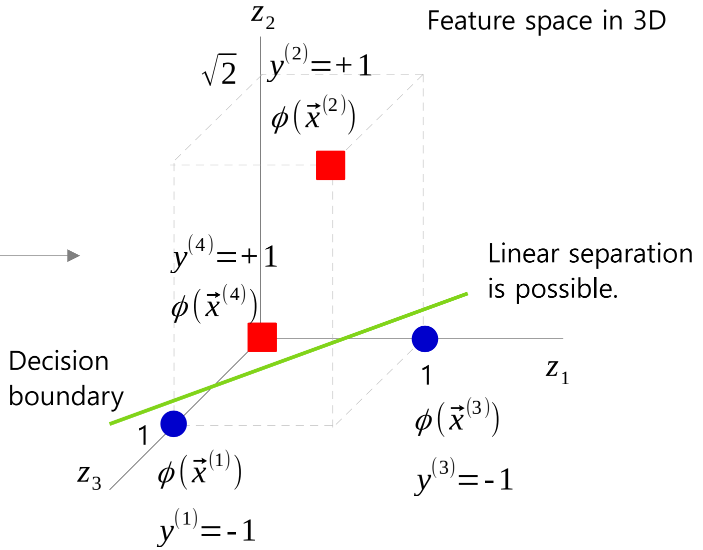
Feature space transformation →

$$\phi(x^{(1)})=(0,0,1)$$

$$\phi(x^{(2)})=(1,\sqrt{2},1)$$

$$\phi(x^{(3)})=(1,0,0)$$

$$\phi(x^{(4)})=(0,0,0)$$



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

We can find a decision boundary in this space.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Non-linear decision boundary

- 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.

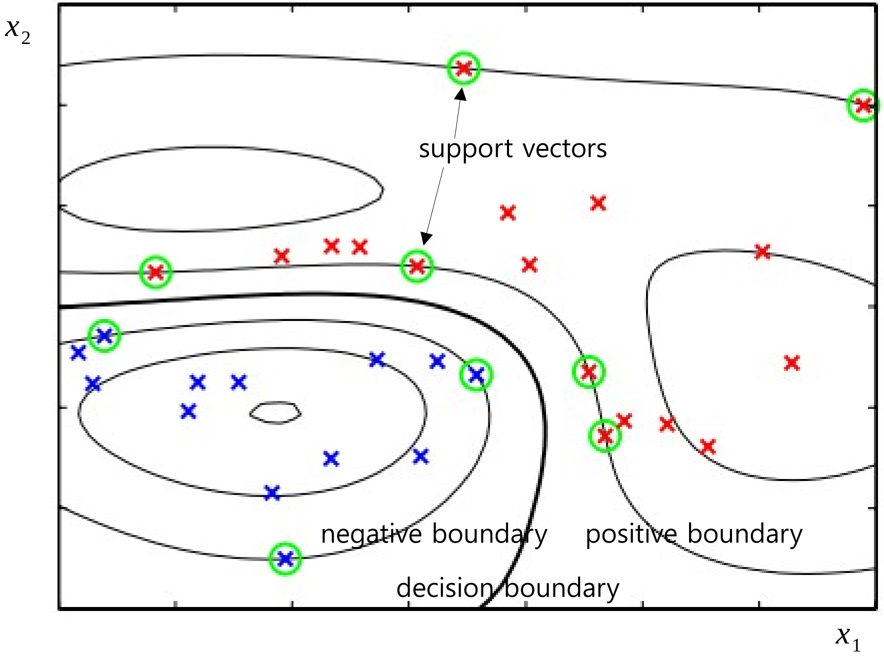


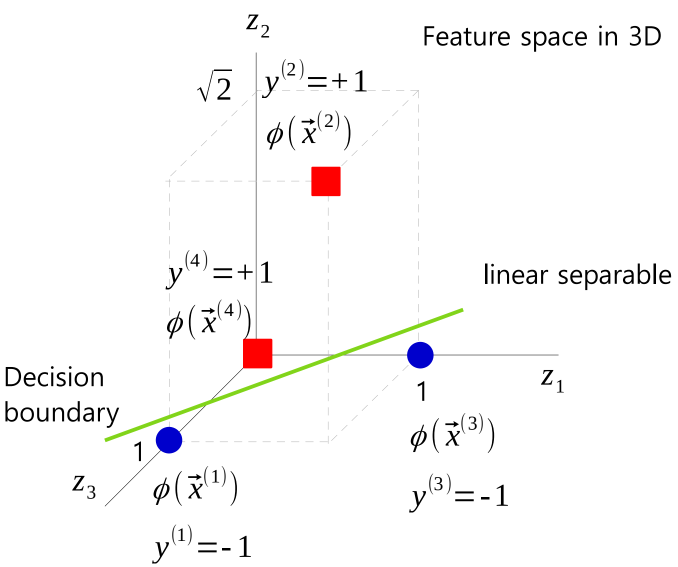
Figure 7.2
Example of synthetic data from two classes in two dimensions showing contours of constant y(x) obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Primal, Dual Lagrange

- 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.



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

- Primal Lagrange

These values are unknown.

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

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

- Dual Lagrange

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

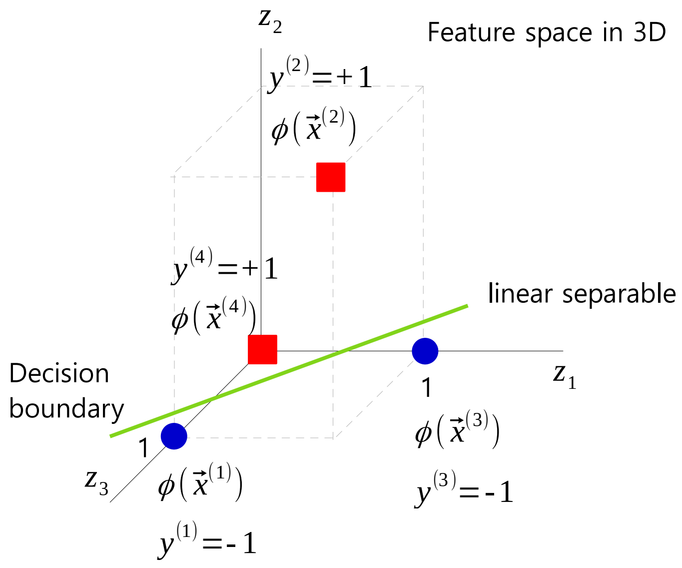
Since we can find this value using a kernel function, we can solve QP to find λ and find b as before. Since we do not know ϕ(x), we cannot find w.

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

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Decision function



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

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

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

- Find b using the support vectors (SV).

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

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 $x^{(i)}$, <u>a numerically more stable solution</u> is obtained by first multiplying through by $y^{(i)}$, making use of $y^{(i)2}$ = 1, and then <u>averaging these equations over all support vectors</u> and solving for b to give (source: Bishop, Pattern Recognition and Machine Learning, p.330, equation 7.18)

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

for linear SVM:

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

(i: support vectors)

- Decision function

test data

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

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

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

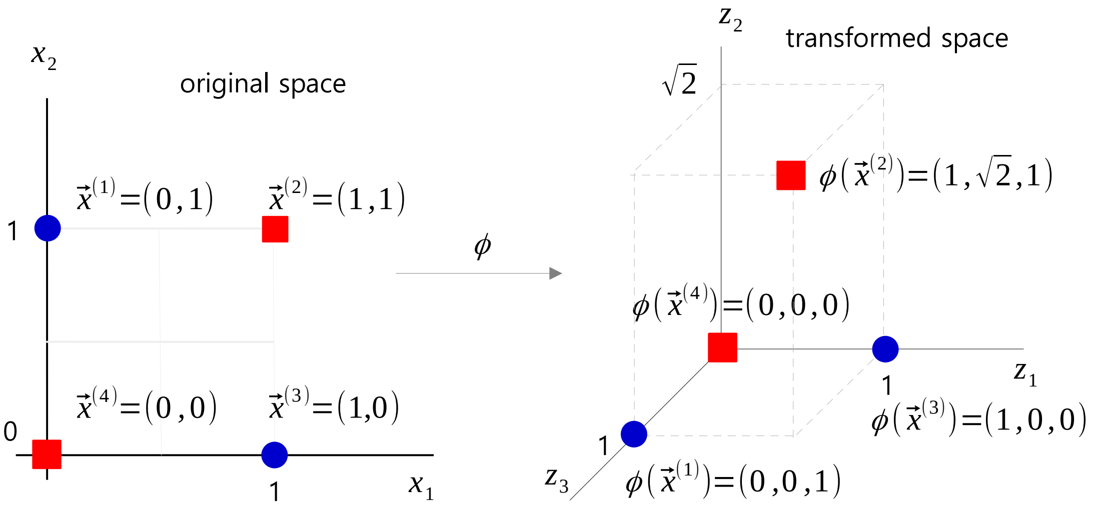
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Kernel trick – finding $\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})$

- Even if you don't know ϕ(x), you can determine the decision function as long as you know $\phi(x_i)\cdot\phi(x_j)$.
- Using an appropriate kernel function, $\phi(x_i)\cdot\phi(x_j)$ can be obtained from the original data x_i and x_j.
- Polynomial, Gaussian, Sigmoid, etc. can be used as kernel functions.

$$\phi\ :\ \vec{x}=(x_1,\ x_2)\ \rightarrow\ \phi(\vec{x})=(x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2)$$



- As shown on the left, if we know ϕ(x), we can get $\phi(x_i)\cdot\phi(x_j)$.
- In reality, we don't know the ϕ(x).

$$\phi(\vec{x}^{(1)})\cdot\phi(\vec{x}^{(2)})=(0,0,1)\cdot(1,\sqrt{2},1)^T=1$$

$$\phi(\vec{x}^{(1)})\cdot\phi(\vec{x}^{(3)})=(0,0,1)\cdot(1,0,0)^T\quad=0$$

$$\phi(\vec{x}^{(2)})\cdot\phi(\vec{x}^{(2)})=(1,\sqrt{2},1)\cdot(1,\sqrt{2},1)^T=4$$

- For example, let's use a quadratic polynomial as the kernel function.

$$\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})\ \stackrel{\text{def}}{=}\ (\vec{x}^{(i)}\cdot\vec{x}^{(j)})^2$$

- Results of calculating $\phi(x_i)\cdot\phi(x_j)$ using the above kernel function. You can see that the results are the same as above. In other words, you can find $\phi(x_i)\cdot\phi(x_j)$ without knowing ϕ itself.

$$\phi(\vec{x}^{(1)})\cdot\phi(\vec{x}^{(2)})=(\vec{x}^{(1)}\cdot\vec{x}^{(2)})^2=1$$

$$\phi(\vec{x}^{(1)})\cdot\phi(\vec{x}^{(3)})=(\vec{x}^{(1)}\cdot\vec{x}^{(3)})^2=0$$

$$\phi(\vec{x}^{(2)})\cdot\phi(\vec{x}^{(2)})=(\vec{x}^{(2)}\cdot\vec{x}^{(2)})^2=4$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Kernel functions

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

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

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

$\downarrow \phi$

Polynomial:

$k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y})^p$ ← $\phi(\vec{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)$ (p=2)

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

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

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

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

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

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

$$k(\vec{x},\vec{y})=(x_1^2,\sqrt{2}x_1x_2,x_2^2)\cdot(y_1^2,\sqrt{2}y_1y_2,y_2^2)$$
$$= x_1^2y_1^2+2x_1y_1x_2y_2+x_2^2y_2^2$$
$$= (x_1y_1+x_2y_2)^2=(\vec{x}\cdot\vec{y})^2$$

- 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

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Kernel matrix, Positive Semi-Definite (PSD), Mercer's theorem

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

$$K = \begin{bmatrix} k\left(x^{(1)}, x^{(1)}\right) & k\left(x^{(1)}, x^{(2)}\right) \\ k\left(x^{(2)}, x^{(1)}\right) & k\left(x^{(2)}, x^{(2)}\right) \end{bmatrix} \text{ if n=2.} \qquad k\left(\vec{x}^{(i)}, \vec{x}^{(j)}\right) = k\left(\vec{x}^{(j)}, \vec{x}^{(i)}\right)$$

- 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

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

- Uses eigenvalues to roughly determine whether a given function is a valid kernel function.

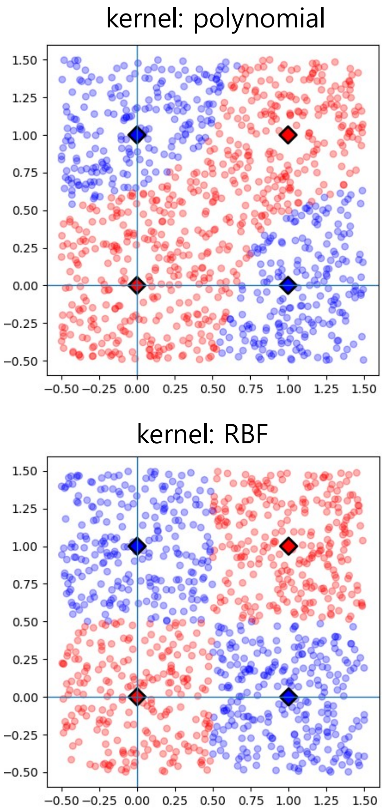
```
# [MXML-6-05] 5.check_kernel.py
# For arbitrary real data, if the eigenvalues of the kernel
# matrix (K) are all non-negative, then K is positive
# semi-definite (PSD) and is a valid kernel function.
import numpy as np

x = np.random.rand(100, 2)  # random dataset (2-dims)
n = x.shape[0]

# kernel functions
rbf_kernel = lambda a, b: np.exp(-np.linalg.norm(a - b)**2 / 2)
pol_kernel = lambda a, b: (1 + np.dot(a, b)) ** 2
sig_kernel = lambda a, b: np.tanh(3 * np.dot(a, b) + 5)
cos_kernel = lambda a, b: np.cos(np.dot(a, b))
kernels = [rbf_kernel, pol_kernel, sig_kernel, cos_kernel]
names = ['RBF', 'Polynomial', 'Sigmoid', 'Cos']

for kernel, name in zip(kernels, names):
    # Kernel matrix (Gram matrix).
    K = np.array([kernel(x[i], x[j])
                  for i in range(n)
                  for j in range(n)]).reshape(n, n)

    # Find eigenvalues, eigenvectors
    w, v = np.linalg.eig(K)

    # The function defined above is a valid kernel if all
    # eigenvalues of K are non-negative.
    print('\nKernel : ' + name)
    print('max eigenvalue =', w.max().round(3))
    print('min eigenvalue =', w.min().round(8))

    if w.min().real > -1e-8:
        print('==> valid kernel')
    else:
        print('==> invalid kernel')
```

```
Kernel : RBF
max eigenvalue = (85.697+0j)
min eigenvalue = (-0+0j)
==> valid kernel

Kernel : Polynomial
max eigenvalue = (246.356+0j)
min eigenvalue = (-0-0j)
==> valid kernel

Kernel : Sigmoid
max eigenvalue = (99.999+0j)
min eigenvalue = (-0.00034159+0j)
==> invalid kernel

Kernel : Cos
max eigenvalue = (86.204+0j)
min eigenvalue = (-8.62055386+0j)
==> invalid kernel
```

K is not a PSD in the strict sense because it is difficult to say that it is non-negative. However, it is widely used as a kernel. (reference : Hsuan-Tien Lin, 2003, A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods)

It's definitely negative. Cos(.) cannot be a kernel function.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai



6. Support Vector Machine (SVM)

Part 6: Non-Linear SVM – Implementation

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Quadratic Programming for nonlinear SVM

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

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

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

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

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

$$\boxed{\begin{aligned} & L_D=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\lambda^T\cdot H\cdot\lambda \\ & \text{s.t}\quad 0\le\lambda_i\le C \\ & \sum_{i=1}^{N}\lambda_i y^{(i)}=0 \end{aligned}}$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Quadratic Programming for nonlinear SVM

- Standard form of QP

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

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

$$Ax = b$$

- Lagrange dual function for nonlinear SVM

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

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

$$\lambda_i \le C$$

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

← 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`

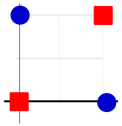
$A\ :=\ y$ `size = N x 1`

$b\ :=\ 0$ `scalar`

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

MX-AI

- Implement nonlinear SVM from scratch using CVXOPT.



```python
# [MXML-6-06] 6.cvxopt(kernel_trick).py
# Implemen nonlinear SVM using CVXOPT
import numpy as np
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
import matplotlib.pyplot as plt

# 4 data samples. 2 '+' samples, 2 '-' samples
x = np.array([[0., 1.], [1., 1.], [1., 0.], [0., 0.]])
y = np.array([[-1.], [1.], [-1.], [1.]])

# kernel function
def kernel(a, b, p=3, r=0.5, type="rbf"):
    if k_type == "poly":
        return (1 + np.dot(a, b)) ** p
    else:
        return np.exp(-r * np.linalg.norm(a - b)**2)

C = 1.0            # regularization constant
N = x.shape[0]     # the number of data points
k_type = "poly"    # kernel type: poly or rbf

# Kernel matrix. k(xi, xj) = ϕ(xi)ϕ(xj).
K = np.array([kernel(x[i], x[j], type=k_type)
        for i in range(N)
        for j in range(N)]).reshape(N, N)

# Construct matrices required for QP in standard form.
H = np.outer(y, y) * K
P = cvxopt_matrix(H)
q = cvxopt_matrix(np.ones(N) * -1)
A = cvxopt_matrix(y.reshape(1, -1))
```

```python
b = cvxopt_matrix(np.zeros(1))
g = np.vstack([-np.eye(N), np.eye(N)])
G = cvxopt_matrix(g)
h1 = np.hstack([np.zeros(N), np.ones(N) * C])
h = cvxopt_matrix(h1)

# solver parameters
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Perform QP
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
lamb = np.array(sol['x']) # the solution to the QP, λ

# Find support vectors
sv_i = np.where(lamb > 1e-5)[0]
sv_m = lamb[sv_i]    # lambda
sv_x = x[sv_i]
sv_y = y[sv_i]

# Calculate b using the support vectors and calculate the average.
def cal_wphi(cond):
    wphi = []
    idx = np.where(cond)[0]
    for i in idx:
        wp = [sv_m[j] * sv_y[j] * kernel(sv_x[i], sv_x[j], type=k_type)\
              for j in range(sv_x.shape[0])]
        wphi.append(np.sum(wp))
    return wphi

b = -(np.max(cal_wphi(sv_y > 0)) + np.min(cal_wphi(sv_y < 0))) / 2.
```

$$k(\vec{x},\vec{y})=(\vec{x}\cdot\vec{y}+c)^p,\ (c>0)$$

$$k(\vec{x},\vec{y})=\exp(-\gamma\|\vec{x}-\vec{y}\|^2)$$

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

$$\vec{w}\cdot\phi(\vec{x})=\sum_{s\in SV}\lambda_s y^{(s)}\phi(\vec{x}^{(s)})\cdot\phi(\vec{x})$$

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

$$b=-\frac{max\left(\vec{w}\cdot\phi(\vec{x}_{SV}^{(+)})\right)+min\left(\vec{w}\cdot\phi(\vec{x}_{SV}^{(-)})\right)}{2}$$

MX-AI

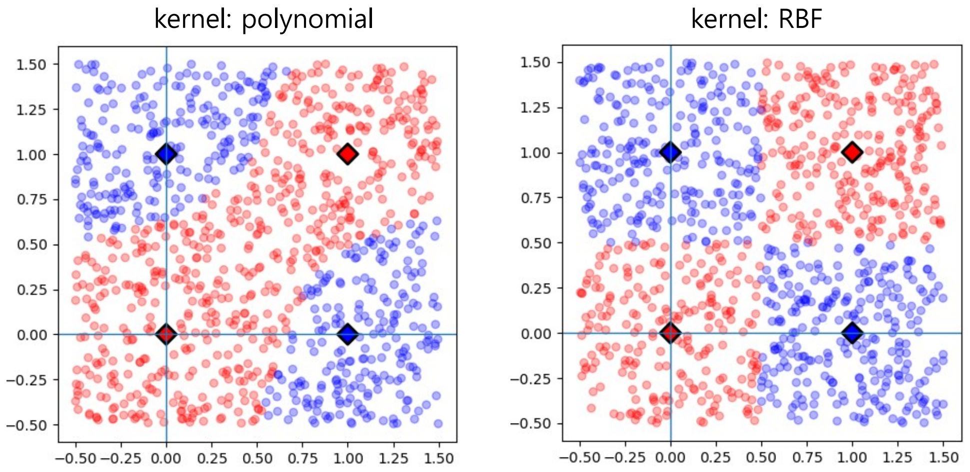
- Implement nonlinear SVM from scratch using CVXOPT.

```
# Predict the class of test data.
x_test = np.random.uniform(-0.5, 1.5, (1000, 2))
n_test = x_test.shape[0]
n_sv = sv_x.shape[0]
ts_K = np.array([kernel(sv_x[i], x_test[j], type=k_type)
        for i in range(n_sv)
        for j in range(n_test)]).reshape(n_sv, n_test)
```

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

```
# decision function
y_hat = np.sum(sv_m * sv_y * ts_K, axis=0).reshape(-1, 1) + b
y_pred = np.sign(y_hat)

# Visualize test data and classes.
plt.figure(figsize=(5,5))
test_c = ['red' if a == 1 else 'blue' for a in y_pred]
sv_c = ['red' if a == 1 else 'blue' for a in sv_y]
plt.scatter(x_test[:, 0], x_test[:, 1], s=30, c=test_c,
            alpha=0.3)
plt.scatter(sv_x[:, 0], sv_x[:, 1], s=100, marker='D', c=sv_c,
            ec='black', lw=2)
plt.axhline(y=0, lw=1)
plt.axvline(x=0, lw=1)
plt.show()
```

```
     pcost       dcost       gap    pres   dres
 0: -1.1967e+00 -5.9612e+00  5e+00  1e-16  1e-15
 1: -1.2227e+00 -1.4182e+00  2e-01  2e-16  4e-16
 2: -1.2376e+00 -1.2477e+00  1e-02  2e-16  3e-16
 3: -1.2381e+00 -1.2384e+00  3e-04  2e-16  3e-16
 4: -1.2381e+00 -1.2381e+00  3e-06  2e-16  1e-15
 5: -1.2381e+00 -1.2381e+00  3e-08  1e-16  1e-16
 6: -1.2381e+00 -1.2381e+00  3e-10  1e-16  4e-16
 7: -1.2381e+00 -1.2381e+00  3e-12  2e-16  1e-15
Optimal solution found.
```

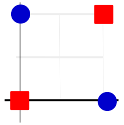


Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

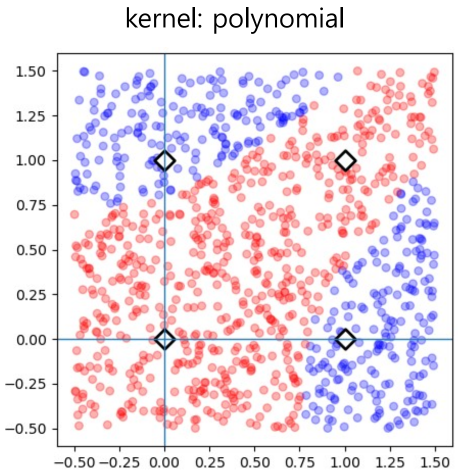
■ Implement nonlinear SVM using SVC.

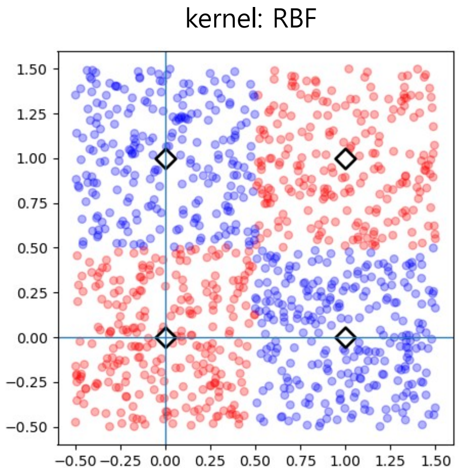
```
# [MXML-6-06] 7.SVC(kernel_trick).py
# Implement nonlinear SVM using SVC.
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt

# 4 data samples. 2 '+' samples, 2 '-' samples
x = np.array([[0., 1.], [1., 1.], [1., 0.], [0., 0.]])
y = np.array([-1., 1., -1., 1.])

C = 1.0
#model = SVC(C=C, kernel='rbf', gamma=0.5)
model = SVC(C=C, kernel='poly', degree=3)
model.fit(x, y)

# Intercept (b)
# w = model.coef_[0]
# AttributeError: coef_ is only available when using a linear kernel
b = model.intercept_[0]

# Predict the class of test data.
x_test = np.random.uniform(-0.5, 1.5, (1000, 2))

# decision function
y_hat = model.decision_function(x_test)
y_pred = np.sign(y_hat)
# y_pred = model.predict(x_test)  # It is the same as above.
```



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

```
# Visualize test data and classes.
plt.figure(figsize=(5,5))
test_c = ['red' if a == 1 else 'blue' for a in y_pred]
plt.scatter(x_test[:, 0], x_test[:, 1], s=30, c=test_c,
            alpha=0.3)
plt.scatter(x[:, 0], x[:, 1], s=100, marker='D', c='white',
            ec='black', lw=2)
plt.axhline(y=0, lw=1)
plt.axvline(x=0, lw=1)
plt.show()
```





Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Classify Titanic dataset using CVXOPT and SVC

```
# [MXML-6-06] 8.Kernel(titanic).py
# Classify Titanic dataset using CVXOPT and SVC
import numpy as np
import pandas as pd
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Read the Titanic data and perform some simple preprocessing.
df = pd.read_csv('data/titanic.csv')
df['Age'].fillna(df['Age'].mean(), inplace = True)
df['Embarked'].fillna('N', inplace = True)
df['Sex'] = df['Sex'].factorize()[0]
df['Embarked'] = df['Embarked'].factorize()[0]
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1,
        inplace=True)

#  Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
# 0       0       3    0  22.0      1      0   7.2500         0
# 1       1       1    1  38.0      1      0  71.2833         1
# 2       1       3    1  26.0      0      0   7.9250         0
# 3       1       1    1  35.0      1      0  53.1000         0
# 4       0       3    0  35.0      0      0   8.0500         0

# Generate training and test data
y = np.array(df['Survived']).reshape(-1,1).astype('float') * 2 - 1
x = np.array(df.drop('Survived', axis=1))
x_train, x_test, y_train, y_test = train_test_split(x, y)
```

```
# Normalize the training and test data
x_mean = x_train.mean(axis=0).reshape(1, -1)
x_std = x_train.std(axis=0).reshape(1, -1)
x_train = (x_train - x_mean) / x_std
x_test = (x_test - x_mean) / x_std

# RBF kernel function
def kernel(a, b, r=0.5):
    return np.exp(-r * np.linalg.norm(a - b)**2)

C = 1.0                 # regularization constant
N = x_train.shape[0]    # the number of data points

# Kernel matrix. k(xi, xj) = φ(xi)φ(xj).
K = np.array([kernel(x_train[i], x_train[j])
        for i in range(N)
        for j in range(N)]).reshape(N, N)

# Construct the matrices required for QP in standard form.
H = np.outer(y_train, y_train) * K
P = cvxopt_matrix(H)
q = cvxopt_matrix(np.ones(N) * -1)
A = cvxopt_matrix(y_train.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))
g = np.vstack([-np.eye(N), np.eye(N)])
G = cvxopt_matrix(g)
h1 = np.hstack([np.zeros(N), np.ones(N) * C])
h = cvxopt_matrix(h1)
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

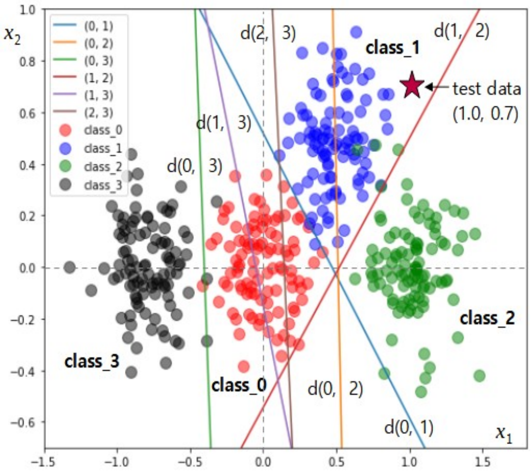
■ Classify Titanic dataset using CVXOPT and SVC

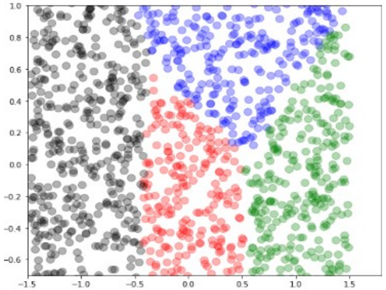
```
# solver parameters
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Perform QP
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
lamb = np.array(sol['x'])  # the solution to the QP, λ

# Find support vectors
sv_i = np.where(lamb > 1e-5)[0]
sv_m = lamb[sv_i]    # lambda
sv_x = x_train[sv_i]
sv_y = y_train[sv_i]

# Calculate b using the support vectors and calculate the average.
def cal_wphi(cond):
    wphi = []
    idx = np.where(cond)[0]
    for i in idx:
        wp = [sv_m[j] * sv_y[j] * kernel(sv_x[i], sv_x[j]) \
              for j in range(sv_x.shape[0])]
        wphi.append(np.sum(wp))
    return wphi

b = -(np.max(cal_wphi(sv_y > 0)) + np.min(cal_wphi(sv_y < 0))) / 2.

# Predict the class of test data.
n_test = x_test.shape[0]
n_sv = sv_x.shape[0]
```

```
ts_K = np.array([kernel(sv_x[i], x_test[j])
        for i in range(n_sv)
        for j in range(n_test)]).reshape(n_sv, n_test)

# decision function
y_hat = np.sum(sv_m * sv_y * ts_K, axis=0).reshape(-1, 1) + b
y_pred = np.sign(y_hat)
acc = (y_pred == y_test).mean()
print('\nCVXOPT: The accuracy of the test data= {:.4f}'\
       .format(acc))

# Compare with sklearn's SVC results.
from sklearn.svm import SVC
model = SVC(C=C, kernel='rbf', gamma=0.5)
model.fit(x_train, y_train.reshape(-1,))
y_pred = model.predict(x_test)

acc = (y_pred == y_test.reshape(-1,)).mean()
print('   SVC: The accuracy of the test data= {:.4f}'\
      .format(acc))
```

```
     pcost       dcost       gap    pres   dres
 0: -2.7828e+02 -1.6738e+03  8e+03  3e+00  2e-15
 1: -1.8865e+02 -1.0903e+03  1e+03  9e-02  2e-15
 ...
18: -2.2825e+02 -2.2825e+02  3e-10  9e-16  2e-15
Optimal solution found.

CVXOPT: The accuracy of the test data = 0.8072
   SVC: The accuracy of the test data = 0.8072
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai





6. Support Vector Machine

(SVM)

Part 7: Multiclass Classification - One-vs-One

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Multiclass classification : One-vs-One (OvO), One-vs-Rest (OvR) = One-vs-All (OvA)

- SVM is a mathematical model for binary classification. Multiclass classification requires performing binary classification multiple times.
- There are two methods for multiclass classification: One-vs-One (OvO), One-vs-Rest (OvR), or One-vs-All (OvA).

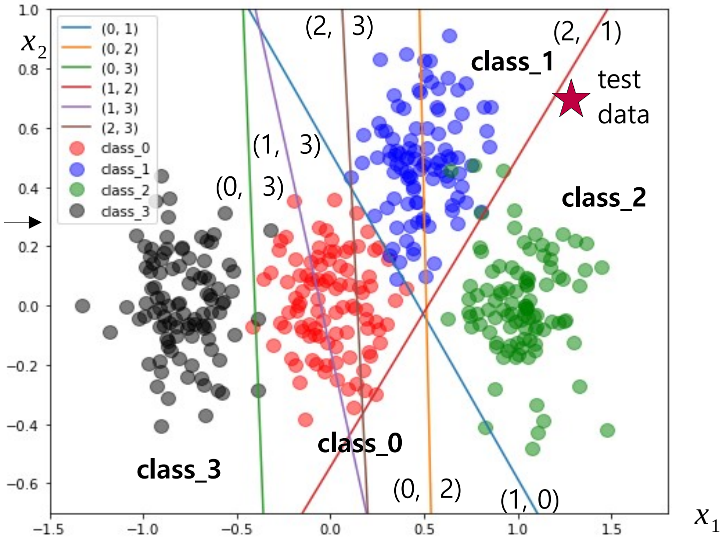
- **One-vs-One (OvO)**

  - Select two classes to find the boundary.
  - A total of ${}_nC_2$ boundaries are created. m = n * (n-1) / 2, (n: the number of classes, m: the number of boundaries)
  - Create m boundaries using the training data and use these boundaries to classify the test data. If there are many classes, it takes a long time because many boundaries need to be created, but the results are stable.

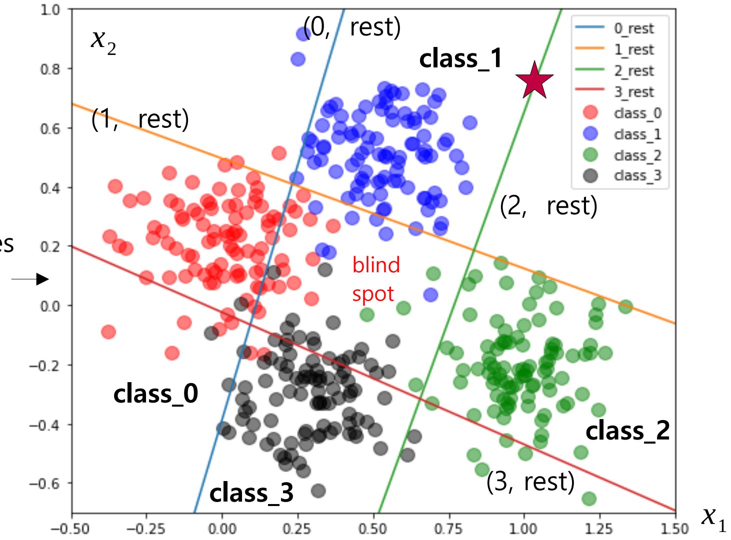
- **One-vs-Rest (OvR) or One-vs-All (OvA)**

  - Find the boundary between one class and the rest.
  - A total of n boundaries are created. m = n, (n: the number of classes, m: the number of boundaries)
  - Create m boundaries using the training data and use these boundaries to classify the test data. Compared to OvO, it has fewer boundaries and faster learning speed, but may have blind spots and is less stable.

Six boundaries divide the four classes.



Four boundaries divide the four classes.

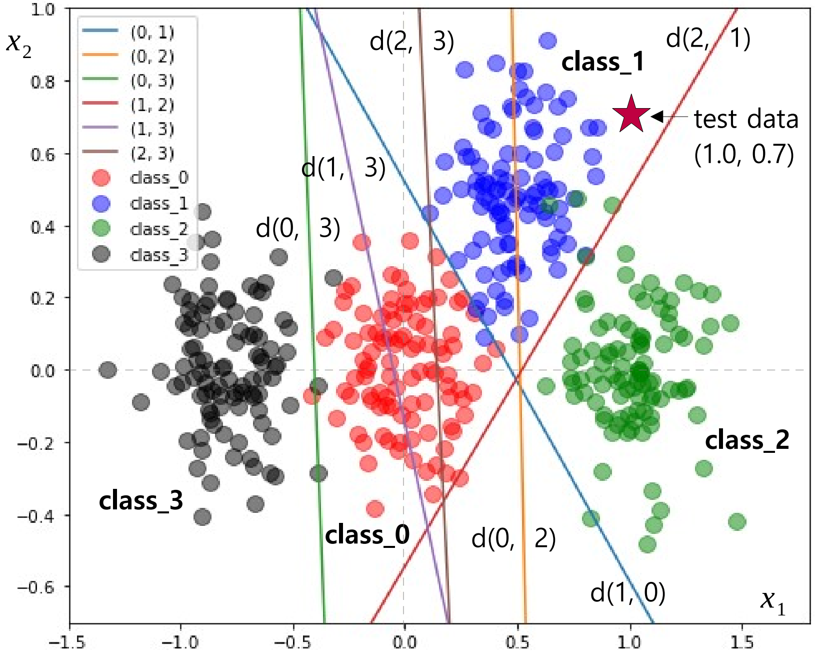


Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

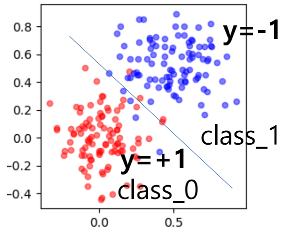
MX-AI

One-vs-One (OvO)

- Classify the classes of the test data using six boundaries. (majority voting)




```
                 w                b
d(0, 1) : [-3.6 , -3.03]      [1.74]
d(0, 2) : [-3.72, -0.41]      [1.83]
d(0, 3) : [ 4.62,  0.73]      [1.8 ]
d(1, 2) : [-3.63,  3.56]      [1.82]
d(1, 3) : [ 2.46,  0.89]      [0.19]
d(2, 3) : [ 1.84,  0.21]      [-0.2]
```



The decision functions below are obtained by defining y as shown in this figure.

Decision boundary	Decision function	Test data: (1.0, 0.7)
$d(0,1): -3.6\,x_1 - 3.0\,x_2 + 1.74 = 0$	$\hat{y} = -3.6\,x_1 - 3.0\,x_2 + 1.74$	$\hat{y} = -3.96$
$d(0,2): -3.72\,x_1 - 0.41\,x_2 + 1.83 = 0$	$\hat{y} = -3.72\,x_1 - 0.41\,x_2 + 1.83$	$\hat{y} = -2.18$
$d(0,3): 4.62\,x_1 + 0.73\,x_2 + 1.8 = 0$	$\hat{y} = 4.62\,x_1 + 0.73\,x_2 + 1.8$	$\hat{y} = 6.93$
$d(1,2): -3.63\,x_1 + 3.56\,x_2 + 1.82 = 0$	$\hat{y} = -3.63\,x_1 + 3.56\,x_2 + 1.82$	$\hat{y} = 0.68$
$d(1,3): 2.46\,x_1 + 0.89\,x_2 + 0.19 = 0$	$\hat{y} = 2.46\,x_1 + 0.89\,x_2 + 0.19$	$\hat{y} = 3.27$
$d(2,3): 1.84\,x_1 + 0.21\,x_2 - 0.2 = 0$	$\hat{y} = 1.84\,x_1 + 0.21\,x_2 - 0.2$	$\hat{y} = 1.79$

For d(0,1), the test data is to the right of the boundary, so it is classified as class_1.
If y_hat of d(0,1) > 0, it is classified as class_0, otherwise it is classified as class_1.

* **Decision rule**: if the y-hat of d(A, B) > 0, then class=A, else class = B

$\hat{y}$

d(0, 1) = -3.96 < 0 → class = 1
d(0, 2) = -2.18 < 0 → class = 2
d(0, 3) = 6.93 > 0 → class = 0
d(1, 2) = 0.68 > 0 → class = 1
d(1, 3) = 3.27 > 0 → class = 1
d(2, 3) = 1.79 > 0 → class = 2

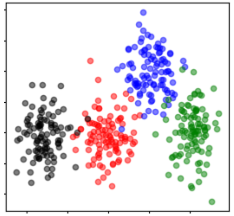
Among the classes found with six boundaries, "1" is the most common, so it is classified as class = 1 by majority vote.

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implement multiclass classification of SVM by One versus One (OvO)

```
# [MXML-6-07] 9.multiclass(OvO).py
# Implement multiclass classification of SVM by One-vs-One (OvO)
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from itertools import combinations

# Generate the data with 4 clusters.
x, y = make_blobs(n_samples=400, n_features=2,
                  centers=[[0., 0.], [0.5, 0.5], [1., 0.], [-0.8, 0.]],
                  cluster_std=0.17)

# Linear SVM
C = 1.0
model = SVC(C=C, kernel='linear', decision_function_shape='ovo')
model.fit(x, y)

w = model.coef_
b = model.intercept_
print("w:\n ", w.round(3))      # shape=(6,2)
print("\nb:\n ", b.round(3))    # shape=(6,)

# w:
#   [[-3.124 -3.497]
#    [-3.583  0.158]
#    [ 4.445  0.104]
#    [-3.742  2.982]
#    [ 2.346  0.941]
#    [ 1.857  0.089]]
# b:
#    [ 1.525  1.948  1.853  2.195  0.038 -0.291]
```



$$\hat{y} = \vec{w}\cdot\vec{x} + b$$

```
# Visualize the data and six boundaries.
plt.figure(figsize=(8,7))
colors = ['red', 'blue', 'green', 'black']
y_color= [colors[a] for a in y]
for label in model.classes_:
    idx = np.where(y == label)
    plt.scatter(x[idx, 0], x[idx, 1], s=100, c=colors[label],
                alpha=0.5, label='class_' + str(label))

# Visualize six boundaries.
comb = list(combinations(model.classes_, 2))
x1_dec = np.linspace(-2.0, 2.0, 50).reshape(-1, 1)
for i in range(w.shape[0]):
    x2_dec = -(w[i, 0] * x1_dec + b[i]) / w[i, 1]
    plt.plot(x1_dec, x2_dec, label=str(comb[i]))
plt.xlim(-1.5, 1.8)
plt.ylim(-0.7, 1.)
plt.legend()
plt.show()
```

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

```
# Predict the classes of the test data.
x_test = np.random.uniform(-1.5, 1.5, (2000, 2))
y_pred1 = model.predict(x_test)

# To understand how OvO works, let's manually implement the
# process of model.predict(x_test). df.shape = (2000, 6)
df = np.dot(x_test, w.T) + b          # decision function
# df = model.decision_function(x_test)  # same as above
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implement multiclass classification of SVM by One versus One (OvO)

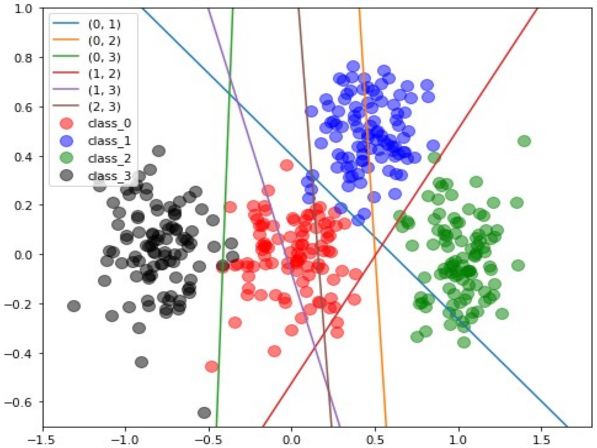
comb (0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)

df [-4.75, -2.58, 6.36, 0.49, 3.4 , 1.77],
[7.57, 7.04, -4.94, 6.75, -3.61, -2.92],
...,
[8.25, 6.97, -4.49, 5.45, -3.65, -2.8],

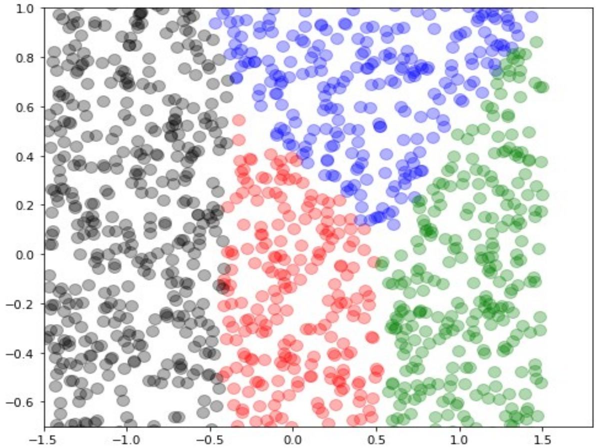
```
classes = model.classes_
n_class = classes.shape[0]

y_pred = []
for i in range(df.shape[0]):
    votes = np.zeros(n_class)
    for j in range(df.shape[1]):  # the number of boundaries
        # if df(i, j) > 0, then class=i, else class=j
        if df[i][j] > 0:
            votes[comb[j][0]] += 1
        else:
            votes[comb[j][1]] += 1

    v = np.argmax(votes)        # majority vote
    y_pred.append(classes[v])
y_pred2 = np.array(y_pred)

# Compare the results of y_pred1 and y_pred2.
if (y_pred1 != y_pred2).sum() == 0:
    print("# y_pred1 and y_pred2 are exactly the same."
else:
    print("# y_pred1 and y_pred2 are not the same.")

# Visualize test data and y_pred1
plt.figure(figsize=(8,7))
y_color= [colors[a] for a in y_pred1]
```

```
for label in model.classes_:
    idx = np.where(y_pred1 == label)
    plt.scatter(x_test[idx, 0], x_test[idx, 1], s=100,
                c=colors[label],
                alpha=0.3, label='class_' + str(label))
plt.xlim(-1.5, 1.8)
plt.ylim(-0.7, 1.)
plt.show()

# y_pred1 and y_pred2 are exactly the same.
```

- Training data and 6 boundaries

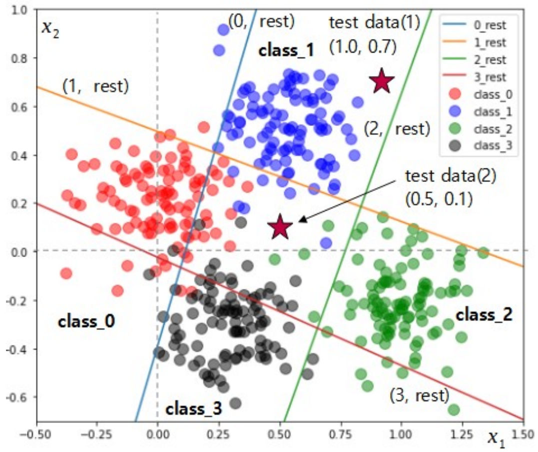


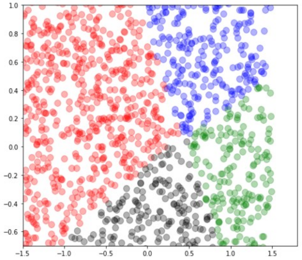
- Test data and decision boundaries



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI





6. Support Vector Machine (SVM)

Part 8: Multiclass Classification - One-vs-Rest

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

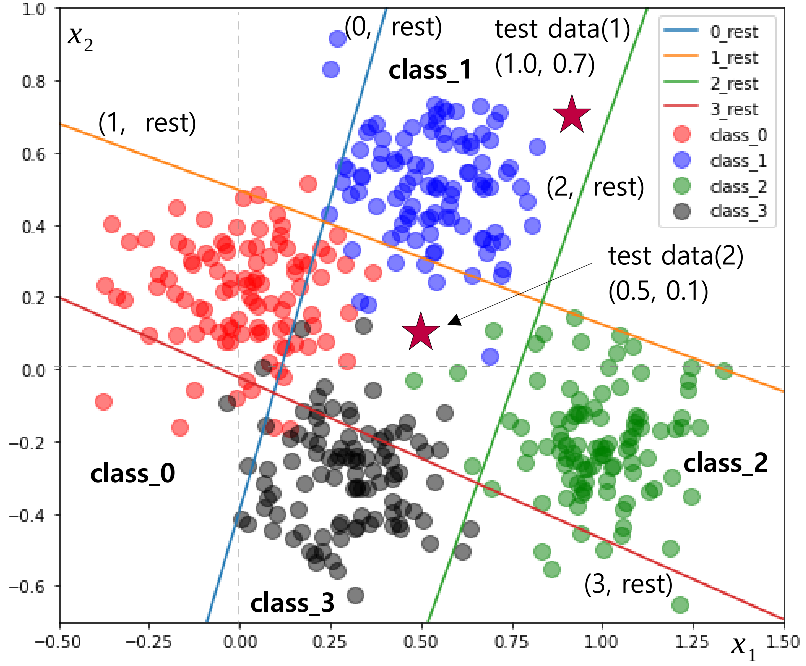
www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ One-vs-Rest (OvR)

- Among multiple boundaries, classification is performed using the boundary with the largest decision function value.




```
                w                 b
0-rest: [-5.87,  1.71]    [ 0.67]
1-rest: [ 2.13,  5.73]    [-2.83]
2-rest: [ 4.88, -1.74]    [-3.76]
3-rest: [-2.59, -5.82]    [-0.14]
```

▪ Decision boundary	▪ Decision function	▪ Test data (1) : (1.0, 0.7)
$d(0, rest): -5.87\,x_1 + 1.71\,x_2 + 0.67 = 0$	$\hat{y} = -5.87\,x_1 + 1.71\,x_2 + 0.67$	$\hat{y} = -4.00$
$d(1, rest): 2.13\,x_1 + 5.73\,x_2 - 2.83 = 0$	$\hat{y} = 2.13\,x_1 + 5.73\,x_2 - 2.83$	$\hat{y} = 3.31$
$d(2, rest): 4.88\,x_1 - 1.74\,x_2 - 3.76 = 0$	$\hat{y} = 4.88\,x_1 - 1.74\,x_2 - 3.76$	$\hat{y} = -0.10$
$d(3, rest): -2.59\,x_1 - 5.82\,x_2 - 0.14 = 0$	$\hat{y} = -2.59\,x_1 - 5.82\,x_2 - 0.14$	$\hat{y} = -6.80$

* **OvO Decision rule**: if the y-hat of d(A, B) > 0, then class=A, else class = B

test data (1) : x = (1.0, 0.7)

d(0, rest) = -4.00 → rest → "It is not class 0"
d(1, rest) = 3.31 → 1 → "It is class 1"
d(2, rest) = -0.10 → rest → "It is not class 2"
d(3, rest) = -6.80 → rest → "It is not class 3"

This test data point can be classified as class 1. The d(1, rest) value is positive and the largest. It can have multiple positive values, and even so, the largest d value can be used to classify.

test data (2) : x = (0.5, 0.1)

d(0, rest) = -2.09 → "It is not class 0"
d(1, rest) = -1.19 → "It is not class 1"
d(2, rest) = -1.49 → "It is not class 2"
d(3, rest) = -2.02 → "It is not class 3"

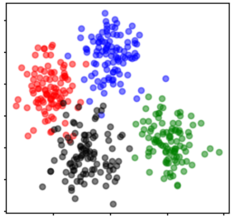
The test data is in a blind spot. The d(1, rest) value is the largest. This means that it is not-class-1, but because the test data is closest to this boundary, it means that it is least likely to be not-class-1. Therefore, it is reasonable to classify it as class 1.

* **OvR Decision rule** : Classify the test data using the boundary with the highest decision function value.

MX-AI

■ Implement multiclass classification of SVM by One-Rest (OvR): OneVsRestClassifier

```
# [MXML-6-08] 10.multiclass(OvR).py
# Implement multiclass classification of SVM by One-Rest (OvR)
# Since SVC operates as an OvO internally, we will use
# OneVsRestClassifier.
import numpy as np
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate the data with 4 clusters.
x, y = make_blobs(n_samples=400, n_features=2,
        centers=[[0., 0.2], [0.5, 0.5], [1., -0.2], [0.3, -0.3]],
        cluster_std=0.15)

# Linear SVM model
C = 1.0
model = OneVsRestClassifier(SVC(C=C, kernel='linear'))
model.fit(x, y)

print(model.estimators_)
# [SVC(kernel='linear'),
#  SVC(kernel='linear'),
#  SVC(kernel='linear'),
#  SVC(kernel='linear')]

w = np.array([m.coef_[0] for m in model.estimators_])      # (4,2)
b = np.array([m.intercept_[0] for m in model.estimators_]) # (4,)
```



```
w :
array([[-6.09,  1.55],
       [ 1.95,  5.33],
       [ 5.45, -1.27],
       [-2.23, -5.14]])
b :
array([ 0.63, -2.57, -4.14, -0.25])
```

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

```
# Visualize the data and 4 boundaries.
plt.figure(figsize=(8,7))
colors = ['red', 'blue', 'green', 'black']
y_color= [colors[a] for a in y]
for label in model.classes_:
    idx = np.where(y == label)
    plt.scatter(x[idx, 0], x[idx, 1], s=100, c=colors[label],
                alpha=0.5, label='class_' + str(label))

# Visualize 4 boundaries.
x1_dec = np.linspace(-2.0, 2.0, 50).reshape(-1, 1)
for i in range(w.shape[0]):
    x2_dec = -(w[i, 0] * x1_dec + b[i]) / w[i, 1]
    plt.plot(x1_dec, x2_dec, label=str(i)+'_rest')
plt.xlim(-0.5, 1.5)
plt.ylim(-0.7, 1.)
plt.legend()
plt.show()
```

```
df: (2000, 4)
array([[ -1.83,   6.55,  -2.02,  -9.79],
       [ -0.87, -10.44,  -1.9 ,   7.53],
       [  7.12,  -6.61,  -9.12,   4.88],
       ...,
```

```
# Predict the classes of the test data.
x_test = np.random.uniform(-1.5, 1.5, (2000, 2))
y_pred1 = model.predict(x_test)

# To understand how OvR works, let's manually implement the
# process of model.predict(x_test). df.shape = (2000, 4)
df = np.dot(x_test, w.T) + b            # decision function
#df = model.decision_function(x_test) # same as above
y_pred2 = df.argmax(axis=1)
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

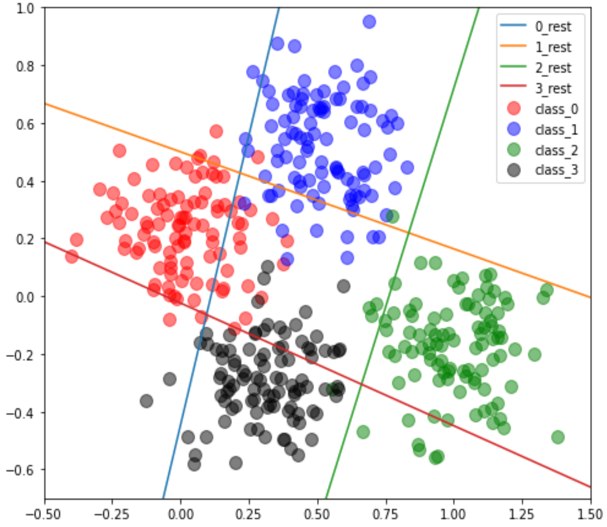
MX-AI

- Implement multiclass classification of SVM by One-Rest (OvR): OneVsRestClassifier

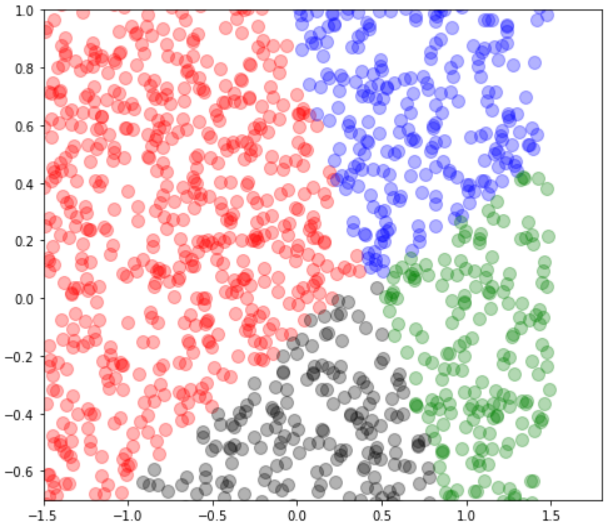
```
# Compare y_pred1 and y_pred2.
if (y_pred1 != y_pred2).sum() == 0:
    print("# y_pred1 and y_pred2 are exactly the same.")
else:
    print("# y_pred1 and y_pred2 are not the same.")

# Visualize test data and y_pred1
plt.figure(figsize=(8,7))
y_color= [colors[a] for a in y_pred1]
for label in model.classes_:
    idx = np.where(y_pred1 == label)
    plt.scatter(x_test[idx, 0], x_test[idx, 1],
                s=100,
                c=colors[label],
                alpha=0.3,
                label='class_' + str(label))

plt.xlim(-1.5, 1.8)
plt.ylim(-0.7, 1.)
plt.show()
```

```
y_pred1 and y_pred2 are exactly the same.
```

- Training data and 4 boundaries



- Test data and decision boundaries



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

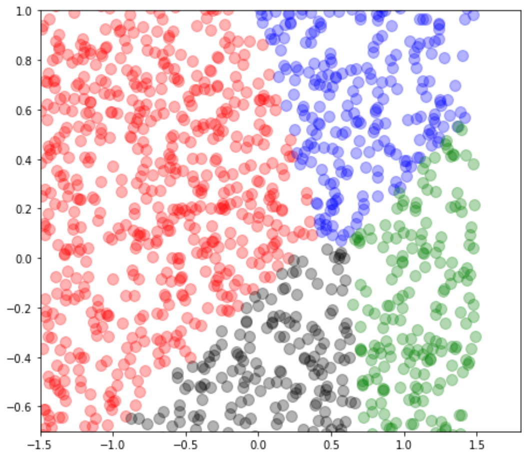
Implement multiclass classification of SVM by One-Rest (OvR): decision_function_shape="ovr" in SVC

- **decision_function_shape{'ovo', 'ovr'}, default='ovr'**
  Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one ('ovo') is always used as a multi-class strategy to train models; an ovr matrix is only constructed from the ovo matrix. The parameter is ignored for binary classification.

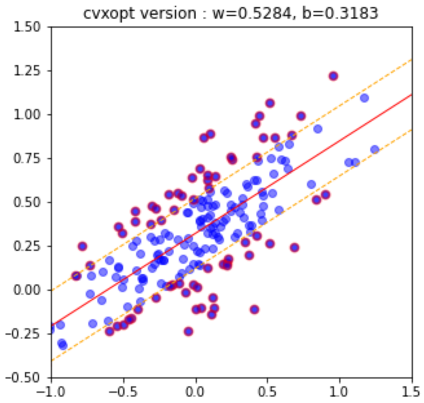
```
# decision_function_shape = 'ovr' in SVC
model2 = SVC(C=C, kernel='linear', decision_function_shape='ovr')
model2.fit(x, y)

# w and b are generated by OvO method.
print("w:\n", model2.coef_)         # (6,2)
print("b:\n", model2.intercept_)   # (6,)

df2 = model2.decision_function(x_test)
y_pred3 = df2.argmax(axis=1)

# Visualize test data and y_pred3
plt.figure(figsize=(8,7))
y_color= [colors[a] for a in y_pred3]
for label in model2.classes_:
    idx = np.where(y_pred3 == label)
    plt.scatter(x_test[idx, 0], x_test[idx, 1], s=100,
                c=colors[label],
                alpha=0.3, label='class_' + str(label))

plt.xlim(-1.5, 1.8)
plt.ylim(-0.7, 1.)
plt.show()
```

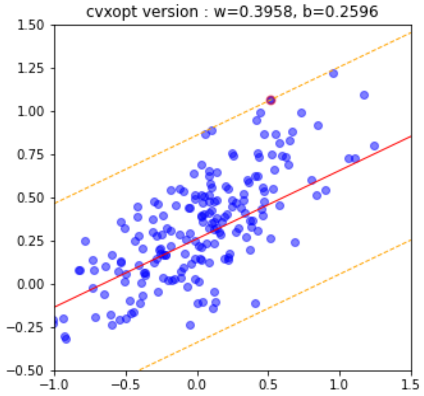
```
w:
 [[-4.94 -1.55]
  [-2.99  0.54]
  [-3.43  3.81]
  [-2.75  2.99]
  [ 1.63  4.17]
  [ 4.61  1.15]]

b:
 [ 1.63  1.41  0.62  1.57 -1.01 -2.86]

df2: (2000, 4)
array([[ 2.14,  3.31,  0.75, -0.31],
       [ 2.19, -0.31,  0.81,  3.31],
       [ 3.31,  0.7 , -0.31,  2.3 ],
       ...,
       [ 3.32,  2.3 , -0.32,  0.71],
       [ 3.32,  2.3 , -0.32,  0.71],
       [ 3.31,  2.28, -0.31,  0.74]])
```

- Test data and decision boundaries



Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai





6. Support Vector Machine (SVM)

Part 9: Linear Support Vector Regression

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

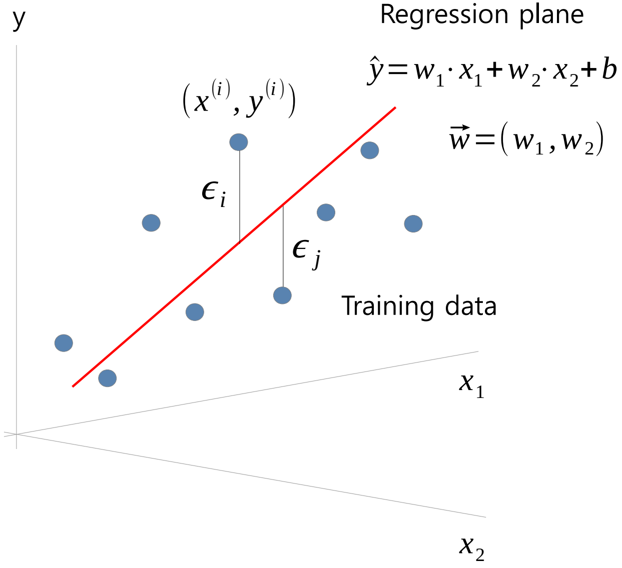
www.youtube.com/@meanxai

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Support Vector Regression (SVR) : Regularized Linear Regression

- Regularized linear regression requires two goals to be considered. One is to minimize the overall errors, and the other is to make w's small. These two goals are a trade-off and must be balanced properly. To achieve both goals, you can use two expressions as follows.



Objective : $$\min_w \Big[\frac{1}{2}\sum_{i=1}^{n} \big(\underbrace{y^{(i)}-\hat{y}^{(i)}}_{\epsilon_i}\big)^2 + \lambda \cdot \|\vec{w}\|^2\Big]$$

main loss: minimize overall errors

penalty term: A penalty is imposed for w's increasing.

another expression $\longrightarrow$

$$\min_w \Big[\frac{1}{2}\|\vec{w}\|^2+\lambda \cdot \sum_{i=1}^{n} \big(\underbrace{y^{(i)}-\hat{y}^{(i)}}_{\epsilon_i}\big)^2\Big]$$

main loss: minimize w's

penalty term: A penalty is imposed for errors increasing.

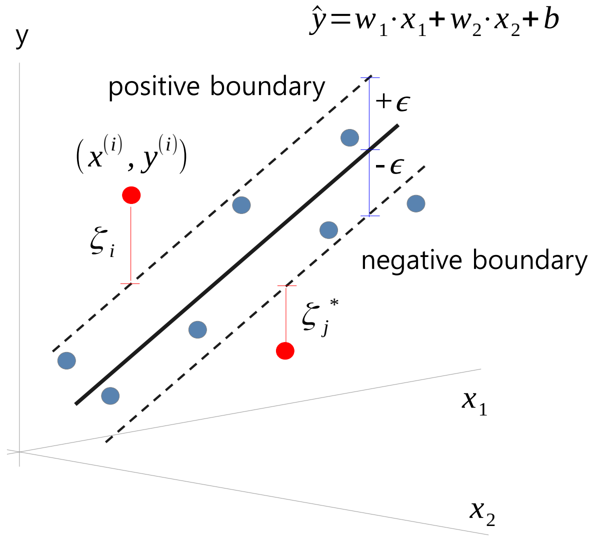
If w is large, even small changes in x will have a large effect on the y estimate. W need to be small so that changes in x do not significantly affect the y estimate. → w's become smaller. → The slope of the regression line decreases.

"Our goal is to find a regression line that has minimal error but is as flat as possible."

"Our goal is to find a regression line that is as flat as possible, but with as small errors as possible."

Objective function for Support Vector regression

- To determine the optimal regression line (plane) for a given ε (error tolerance range), set the objective function and constraints as follows.



$\pm\ \epsilon$: error tolerance range. Data points within this range are considered non-error.

ζ, ζ^* : Error outside error tolerance range. (slack variables).

- **Objective**:

Regularization constant

$$\underset{w,\zeta,\zeta^*}{min}\ [\frac{1}{2}\|\vec{w}\|^2 + C \cdot \sum_{i=1}^{n} (\zeta_i+\zeta_i^*)]$$

main loss

minimize w's

penalty term

A penalty is imposed for errors increasing.

The goal is to create a plane that is as flat as possible for a given ε, while having as small an overall error as possible.

- **Constraints**:

$$y^{(i)}-(\vec{w} \cdot \vec{x}^{(i)}+b) \le \epsilon+\zeta_i$$

$$y^{(i)}-(\vec{w} \cdot \vec{x}^{(i)}-b) \ge -\epsilon-\zeta_i^*$$

$$\zeta_i,\ \zeta_i^* \ge 0$$

MX-AI

Lagrange primal and dual function.

- Constrained optimization problem

$$\min\left[\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}(\zeta_i+\zeta_i^*)\right]$$

s.t.

$$y^{(i)}-\vec{w}\cdot\vec{x}^{(i)}-b-\epsilon-\zeta_i\le 0$$
$$-y^{(i)}+\vec{w}\cdot\vec{x}^{(i)}+b-\epsilon-\zeta_i^*\le 0$$
$$-\zeta_i\le 0$$
$$-\zeta_i^*\le 0$$

- Lagrange primal function

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{n}(\zeta_i+\zeta_i^*)+\sum_{i=1}^{n}(-\eta_i\zeta_i-\eta_i^*\zeta_i^*)+\sum_{i=1}^{n}\lambda_i(y^{(i)}-\vec{w}\cdot\vec{x}^{(i)}-b-\epsilon-\zeta_i)+\sum_{i=1}^{n}\lambda_i^*(-y^{(i)}+\vec{w}\cdot\vec{x}^{(i)}+b-\epsilon-\zeta_i^*)$$

$$(\eta_i,\eta_i^*,\lambda_i,\lambda_i^*\ge 0)$$

$$\frac{\partial L_p}{\partial \vec{w}} = \vec{w}-\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\vec{x}^{(i)}=0 \qquad \frac{\partial L_p}{\partial b} = \sum_{i=1}^{n}(\lambda_i^*-\lambda_i)=0$$

$$\frac{\partial L_p}{\partial \zeta_i} = C-\eta_i-\lambda_i=0 \longrightarrow \lambda_i\le C \qquad \frac{\partial L_p}{\partial \zeta_i^*} = C-\eta_i^*-\lambda_i^*=0 \longrightarrow \lambda_i^*\le C$$

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{n}(\cancel{\zeta_i}+\cancel{\zeta_i^*})+\sum_{i=1}^{n}(-\eta_i\cancel{\zeta_i}-\eta_i^*\cancel{\zeta_i^*})+\sum_{i=1}^{n}\lambda_i(y^{(i)}-\vec{w}\cdot\vec{x}^{(i)}-\cancel{b}-\epsilon-\cancel{\zeta_i})+\sum_{i=1}^{n}\lambda_i^*(-y^{(i)}+\vec{w}\cdot\vec{x}^{(i)}+\cancel{b}-\epsilon-\cancel{\zeta_i^*})$$

$$\sum_{i=1}^{n}(C-\eta_i-\lambda_i)\zeta_i=0$$
$$\sum_{i=1}^{n}(C-\eta_i^*-\lambda_i^*)\zeta_i^*=0$$

- Lagrange dual function

$$L_D=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i-\lambda_i^*)(\lambda_j-\lambda_j^*)\vec{x}^{(i)}\cdot\vec{x}^{(j)}-\epsilon\sum_{i=1}^{n}(\lambda_i+\lambda_i^*)+\sum_{i=1}^{n}y^{(i)}(\lambda_i-\lambda_i^*)$$

s.t.

$$\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)=0$$

$$\left.\begin{matrix}0\le\lambda_i\le C\\ 0\le\lambda_i^*\le C\end{matrix}\right\}\longrightarrow \begin{matrix}\lambda_i\ge 0,\ \lambda\le C\\ \lambda_i^*\ge 0,\ \lambda_i^*\le C\end{matrix}$$

- Decision function

$$\vec{w}=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\vec{x}^{(i)}$$

$\hat{y} = \vec{w}\cdot\vec{x} + b$ ← Decision function (b will be calculated later)

* Source : Alex J. Smola, Bernhard Scholkopf, 1998/2004, A tutorial on support vector regression

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Dual function and Quadratic Programming (QP)

- Dual function

$$L_D = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*)\vec{x}^{(i)}\cdot\vec{x}^{(j)} - \epsilon\sum_{i=1}^{n}(\lambda_i + \lambda_i^*) + \sum_{i=1}^{n} y^{(i)}(\lambda_i - \lambda_i^*)$$

$$\underset{\lambda,\lambda^*}{argmin}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i\lambda_j - \lambda_i\lambda_j^* - \lambda_i^*\lambda_j + \lambda_i^*\lambda_j^*)\vec{x}^{(i)}\cdot\vec{x}^{(j)} + \sum_{i=1}^{n}(\epsilon - y^{(i)})\lambda_i + \sum_{i=1}^{n}(\epsilon + y^{(i)})\lambda_i^*$$

subject to $\sum_{i=1}^{n}(\lambda_i - \lambda_i^*) = 0$

$$-\lambda_i \le 0 \qquad \lambda_i \le C$$
$$-\lambda_i^* \le 0 \qquad \lambda_i^* \le C$$

- Matrices : if n = 2, i = (1, 2), j = (1, 2)

$$K = \begin{bmatrix} \vec{x}^{(1)}\cdot\vec{x}^{(1)} & \vec{x}^{(1)}\cdot\vec{x}^{(2)} \\ \vec{x}^{(2)}\cdot\vec{x}^{(1)} & \vec{x}^{(2)}\cdot\vec{x}^{(2)} \end{bmatrix} \qquad \alpha = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_1^* \\ \lambda_2^* \end{bmatrix} \qquad A = [1,\ 1, -1, -1] \qquad b = 0$$

$$P = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix} \qquad q = \begin{bmatrix} \epsilon - y^{(1)} \\ \epsilon - y^{(2)} \\ \epsilon + y^{(1)} \\ \epsilon + y^{(2)} \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \qquad G\alpha = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_1^* \\ \lambda_2^* \\ -\lambda_1 \\ -\lambda_2 \\ -\lambda_1^* \\ -\lambda_2^* \end{bmatrix} \qquad h = \begin{bmatrix} C \\ C \\ C \\ C \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- Standard form of QP

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

subject to $A\alpha = b$

$$G\alpha \le h$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

Computing b

- Primal function : $L_p = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}(\zeta_i + \zeta_i^*) + \sum_{i=1}^{n}(-\eta_i\zeta_i - \eta_i^*\zeta_i^*) + \sum_{i=1}^{n}\lambda_i(y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} - b - \epsilon - \zeta_i) + \sum_{i=1}^{n}\lambda_i^*(-y^{(i)} + \vec{w}\cdot\vec{x}^{(i)} + b - \epsilon - \zeta_i^*)$

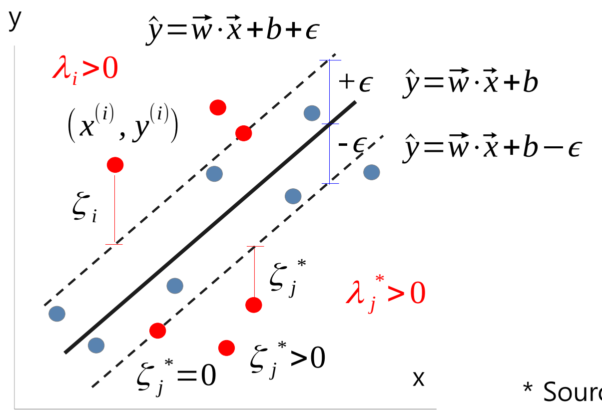
- KKT condition : complementary slackness

1) $\lambda_i(y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} - b - \epsilon - \zeta_i) = 0$ ← If $\lambda_i>0$ and $\zeta_i=0$, the data point is on the positive boundary.

2) $\lambda_i^*(-y^{(i)} + \vec{w}\cdot\vec{x}^{(i)} + b - \epsilon - \zeta_i^*) = 0$ ← If $\lambda_i^*>0$ and $\zeta_i^*=0$, the data point is on the negative boundary.

3) $\eta_i\zeta_i = 0 \rightarrow (C - \lambda_i)\zeta_i = 0$ ← If $\lambda_i=C$, then $\zeta_i>0$ and the data point is outside the positive boundary.

4) $\eta_i^*\zeta_i^* = 0 \rightarrow (C - \lambda_i^*)\zeta_i^* = 0$ ← If $\lambda_i^*=C$, then $\zeta_i^*>0$ and the data point is outside the negative boundary.



- computing b
- $0 \le \lambda_i \le C$ and $0 \le \lambda_i^* \le C$
- In conditions 3) and 4), if λ_i and λ_j^* are less than C, then ζ_i and ζ_i^* must be 0. All of these data points lie between the positive and negative boundaries.
- The data points that satisfy the conditions $0 < \lambda_i < C$ and $0 < \lambda_j^* < C$ have ζ_i and ζ_i^* equal to 0.
- The data points that satisfy conditions 1) and 2) are on the positive and negative boundaries.

1) $0<\lambda_i<C \rightarrow \zeta_i=0 \rightarrow y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} - b - \epsilon = 0$

2) $0<\lambda_i^*<C \rightarrow \zeta_j^*=0 \rightarrow -y^{(i)} + \vec{w}\cdot\vec{x}^{(i)} + b - \epsilon = 0$

1) $b = y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} - \epsilon$: For the data points (x, y) that satisfy the condition $0 < \lambda_i < C$.

2) $b = y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} + \epsilon$: For the data points (x, y) that satisfy the condition $0 < \lambda_i < C$.

* Take the average of b.

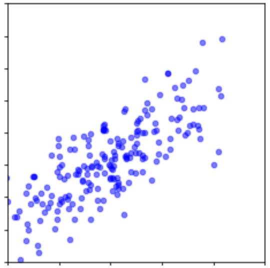
* Source : Alex J. Smola, Bernhard Scholkopf, 1998, A tutorial on support vector regression. 1.4 Computing b, eq. (13)

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implement a Linear SVR from scratch using CVXOPT

```python
# [MXML-6-09] 11.cvxopt(svr_linear).py
import numpy as np
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
import matplotlib.pyplot as plt

# Create a dataset. y = ax + b + noise
def linear_data(a, b, n, s):
   rtn_x = []
   rtn_y = []
   for i in range(n):
       x = np.random.normal(0.0, 0.5)
       y = a * x + b + np.random.normal(0.0, s)
       rtn_x.append(x)
       rtn_y.append(y)

   return np.array(rtn_x).reshape(-1,1),
np.array(rtn_y).reshape(-1,1)

x, y = linear_data(a=0.5, b=0.3, n=200, s=0.2)

eps = 0.2
C = 2.0
n = x.shape[0]

# Construct matrices required for QP in standard form.
K = np.dot(x, x.T)
P = np.hstack([K, -K])
P = np.vstack([P, -P])
q = np.array([[eps]]) + np.vstack([-y, y])
```



```python
A = np.array([1.] * n)
A = np.hstack([A, -A])
b = np.zeros((1,1))
G = np.vstack([-np.eye(2*n), np.eye(2*n)])
h = np.hstack([np.zeros(2*n), np.ones(2*n) * C])

P = cvxopt_matrix(P)
q = cvxopt_matrix(q)
A = cvxopt_matrix(A.reshape(1, -1))
b = cvxopt_matrix(b)
G = cvxopt_matrix(G)
h = cvxopt_matrix(h)

# solver parameters
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Perform QP
sol = cvxopt_solvers.qp(P, q, G, h, A, b)

# λ and λ* are the solution to the dual problem.
lamb = np.array(sol['x'])

# Calculate w using the lambdas, which is the solution to QP.
lamb_i = lamb[:n]     # λ
lamb_s = lamb[n:]     # λ*
w = np.sum((lamb_i - lamb_s) * x, axis=0)
```

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

$$s.t \quad A\alpha = b$$

$$G\alpha \le h$$

$$\alpha = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_1^* \\ \lambda_2^* \end{bmatrix}$$

$$\vec{w} = \sum_{i=1}^{n} (\lambda_i - \lambda_i^*)\vec{x}^{(i)}$$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

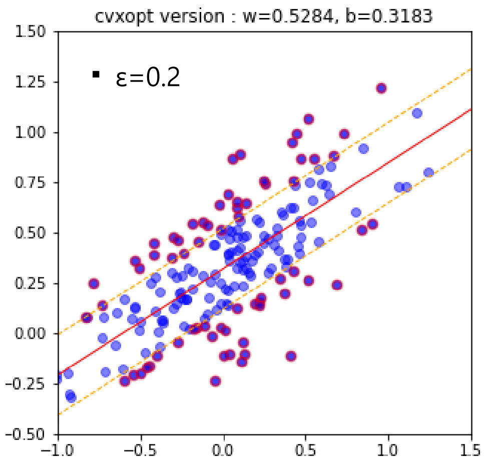
- Implement a Linear SVR from scratch using CVXOPT

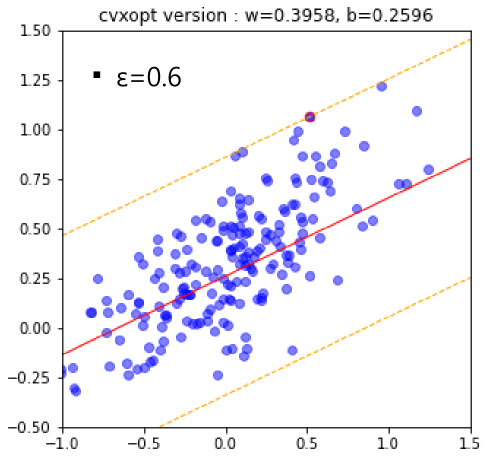
$$b = y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} - \epsilon \quad \leftarrow \; 0<\lambda_i<C$$

$$b = y^{(i)} - \vec{w}\cdot\vec{x}^{(i)} + \epsilon \quad \leftarrow \; 0<\lambda_i^*<C$$

```
# calculating the intercept, b
# [1] Alex Smola, et, al. 1998, A tutorial on support vector
#     regression. 1.4 Computing b, equation - (13)
# [2] Alex Smola, et, al. 2004, A tutorial on support vector
#     regression. 1.4 computing b, equation - (16)

# Calculate b
b = []
# 0 < λ < C
idx_i = np.logical_and(lamb_i > 1e-5, lamb_i < C - 1e-5)
if idx_i.shape[0] > 0:
    b.extend((y[idx_i] - w * x[idx_i] - eps).flatten())

# 0 < λ* < C
idx_s = np.logical_and(lamb_s > 1e-5, lamb_s < C - 1e-5)
if idx_s.shape[0] > 0:
    b.extend((y[idx_s] - w * x[idx_s] + eps).flatten())

if len(b) > 0:
   b = np.mean(b)
   i_sup = np.where(lamb_i > 1e-5)[0]
   s_sup = np.where(lamb_s > 1e-5)[0]

   # Visualize the data and decision function
   plt.figure(figsize=(7,7))
   plt.scatter(x, y, c='blue', alpha=0.5)
   plt.scatter(x[i_sup], y[i_sup], c='blue', ec='red', lw=2.0,
               alpha=0.5)
   plt.scatter(x[s_sup], y[s_sup], c='blue', ec='red', lw=2.0,
               alpha=0.5)
```

```
   x_dec = np.linspace(-1, 1.5, 50).reshape(-1, 1)
   y_dec = w * x_dec + b
   plt.plot(x_dec, y_dec, c='red', lw=1.0)
   plt.plot(x_dec, y_dec + eps, '--', c='orange', lw=1.0)
   plt.plot(x_dec, y_dec - eps, '--', c='orange', lw=1.0)
   plt.xlim(-1, 1.5)
   plt.ylim(-0.5, 1.5)
   plt.title('cvxopt version : w='+str(w[0].round(4))+',
              b='+str(b.round(4)))
   plt.show()
else:
   print('Failed to calculate b.')
```



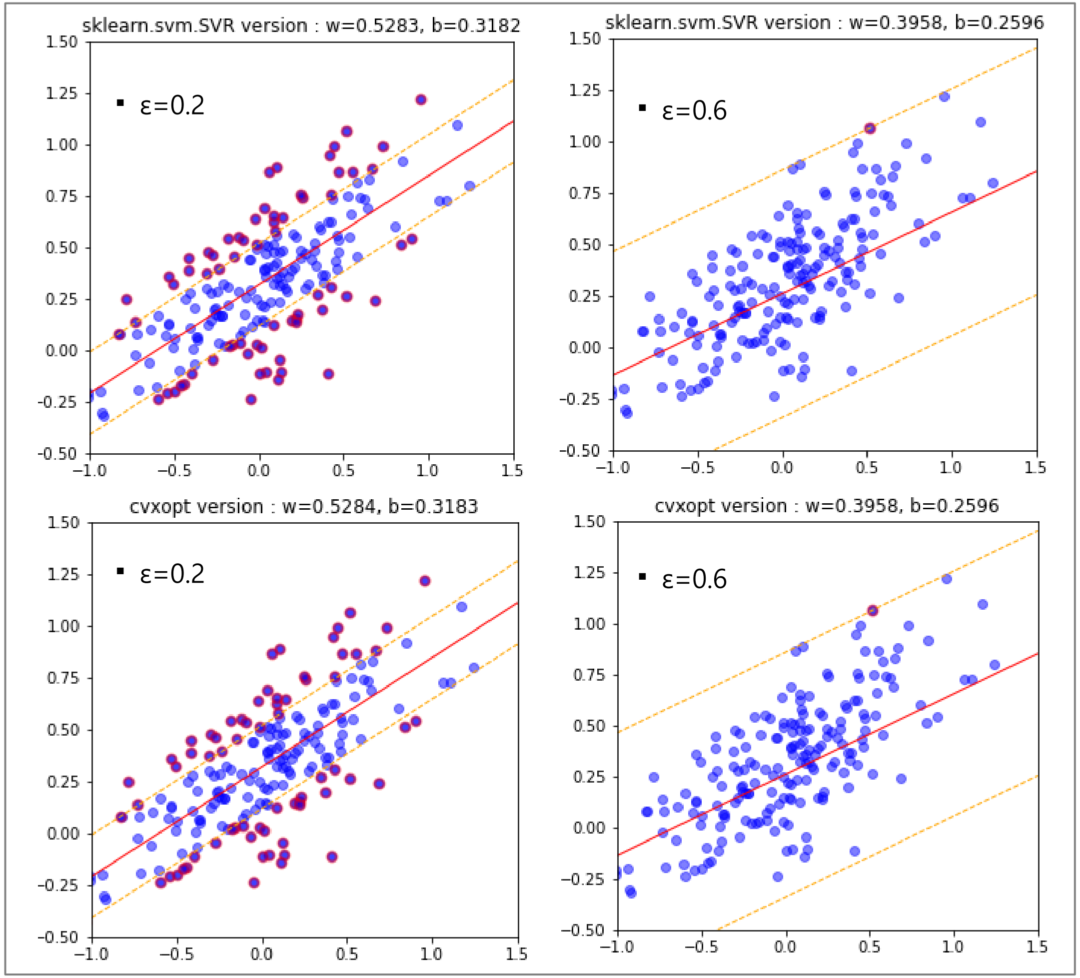


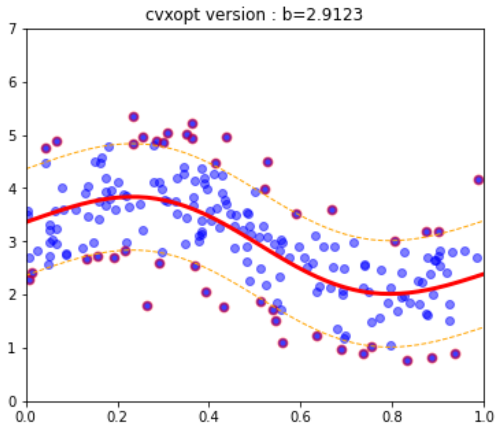
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

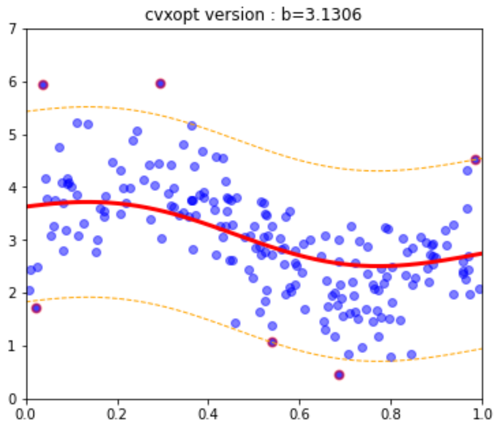
MX-AI

- Implement a Linear SVR using sklearn.svm.SVR

```
# Compare with the results of sklearn.svm.SVR
from sklearn.svm import SVR
model = SVR(C=C, epsilon=eps, kernel='linear')
model.fit(x, y.flatten())

w1 = model.coef_
b1 = model.intercept_
lamb1 = model.dual_coef_
sv_idx = model.support_
sv_x = x[sv_idx]
sv_y = y[sv_idx]

# Visualize the data and decision function
plt.figure(figsize=(7,7))
plt.scatter(x, y, c='blue', alpha=0.5)
plt.scatter(sv_x, sv_y, c='blue', ec='red', lw=2.0, alpha=0.5)
x_dec = np.linspace(-1, 1.5, 50).reshape(-1, 1)
y_dec = w1 * x_dec + b1
plt.plot(x_dec, y_dec, c='red', lw=1.0)
plt.plot(x_dec, y_dec + eps, '--', c='orange', lw=1.0)
plt.plot(x_dec, y_dec - eps, '--', c='orange', lw=1.0)
plt.xlim(-1, 1.5)
plt.ylim(-0.5, 1.5)
plt.title('sklearn.svm.SVR version : w='+str(w1[0][0].round(4))
          +', b='+str(b1[0].round(4)))
plt.show()
```







6. Support Vector Machine (SVM)

Part 10: Nonlinear Support Vector Regression

This video was produced in Korean and translated into English, and the audio was generated by AI (Text-to-Speech).

www.youtube.com/@meanxai

MX-AI

Non-linear regression : Kernel trick

- Primal Lagrange

just replace $\vec{x}^{(i)}$ with $\phi(\vec{x}^{(i)})$

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{n}(\zeta_i+\zeta_i^*)+\sum_{i=1}^{n}(-\eta_i\zeta_i-\eta_i^*\zeta_i^*)+\sum_{i=1}^{n}\lambda_i(y^{(i)}-\vec{w}\cdot\phi(\vec{x}^{(i)})-b-\epsilon-\zeta_i)+\sum_{i=1}^{n}\lambda_i^*(-y^{(i)}+\vec{w}\cdot\phi(\vec{x}^{(i)})+b-\epsilon-\zeta_i^*) \qquad (\eta_i,\eta_i^*,\lambda_i,\lambda_i^*\geq 0)$$

$$\frac{\partial L_p}{\partial \vec{w}}=\vec{w}-\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\phi(\vec{x}^{(i)})=0 \qquad \frac{\partial L_p}{\partial b}=\sum_{i=1}^{n}(\lambda_i^*-\lambda_i)=0 \qquad \frac{\partial L_p}{\partial \zeta_i}=C-\eta_i-\lambda_i=0 \rightarrow \lambda_i\leq C \qquad \frac{\partial L_p}{\partial \zeta_i^*}=C-\eta_i^*-\lambda_i^*=0 \rightarrow \lambda_i^*\leq C$$

$$L_p=\frac{1}{2}\|\vec{w}\|^2+C\sum_{i=1}^{n}(\cancel{\zeta_i}+\cancel{\zeta_i^*})+\sum_{i=1}^{n}(-\eta_i\cancel{\zeta_i}-\eta_i^*\cancel{\zeta_i^*})+\sum_{i=1}^{n}\lambda_i(y^{(i)}-\vec{w}\cdot\phi(\vec{x}^{(i)})-\cancel{b}-\epsilon-\cancel{\zeta_i})+\sum_{i=1}^{n}\lambda_i^*(-y^{(i)}+\vec{w}\cdot\phi(\vec{x}^{(i)})+\cancel{b}-\epsilon-\cancel{\zeta_i^*})$$

- Linear SVR

$$\vec{w}=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\vec{x}^{(i)}$$

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

- Dual Lagrange

$\vec{x}^{(i)}\cdot\vec{x}^{(j)}$ was replaced by $\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})$

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

s.t. $\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)=0$

$$\left.\begin{array}{l}0\leq\lambda_i\leq C\\ 0\leq\lambda_i^*\leq C\end{array}\right\} \rightarrow \begin{array}{l}\lambda_i\geq 0,\ \lambda\leq C\\ \lambda_i^*\geq 0,\ \lambda_i^*\leq C\end{array}$$

- Decision function

$$\vec{w}=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\phi(\vec{x}^{(i)})$$ ← We cannot find w because we don't know $\phi(\vec{x}^{(i)})$.

$$\vec{w}\cdot\phi(\vec{x})=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}) \qquad b = y^{(i)}-\vec{w}\cdot\phi(\vec{x}) \pm \epsilon$$

We can find $\hat{y}$ because we can find $\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})$. →

$$\hat{y}=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}_{test}) + b$$ ← Decision function

$$\hat{y}=\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)k(\vec{x}^{(i)},\vec{x}_{test}) + b$$

The method for calculating b is similar to linear SVM.

* Source : Alex J. Smola, Bernhard Scholkopf, 1998/2004, A tutorial on support vector regression

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Non-linear regression : Quadratic Programming for dual problem

▪ Dual function

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

$\vec{x}^{(i)}\cdot\vec{x}^{(j)}$

$$\underset{\lambda,\lambda^*}{argmin}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i\lambda_j-\lambda_i\lambda_j^*-\lambda_i^*\lambda_j+\lambda_i^*\lambda_j^*)\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})+\sum_{i=1}^{n}(\epsilon-y^{(i)})\lambda_i+\sum_{i=1}^{n}(\epsilon+y^{(i)})\lambda_i^*$$

subject to $\sum_{i=1}^{n}(\lambda_i-\lambda_i^*)=0$

$-\lambda_i\le 0 \qquad \lambda_i\le C$

$-\lambda_i^*\le 0 \qquad \lambda_i^*\le C$

▪ Matrices : if n = 2, i = (1, 2), j = (1, 2)

$$k(\vec{x}^{(i)},\vec{x}^{(j)})=\phi(\vec{x}^{(i)})\cdot\phi(\vec{x}^{(j)})$$

$$K=\begin{bmatrix} k(\vec{x}^{(1)},\vec{x}^{(1)}) & k(\vec{x}^{(1)},\vec{x}^{(2)}) \\ k(\vec{x}^{(2)},\vec{x}^{(1)}) & k(\vec{x}^{(2)},\vec{x}^{(2)}) \end{bmatrix} \qquad \alpha=\begin{bmatrix}\lambda_1\\ \lambda_2\\ \lambda_1^*\\ \lambda_2^*\end{bmatrix}$$

$$A=[1,\ 1,-1,-1] \qquad b=0$$

$$P=\begin{bmatrix} K & -K \\ -K & K \end{bmatrix} \qquad q=\begin{bmatrix}\epsilon-y^{(1)}\\ \epsilon-y^{(2)}\\ \epsilon+y^{(1)}\\ \epsilon+y^{(2)}\end{bmatrix}$$

$$G=\begin{bmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1\\ -1&0&0&0\\ 0&-1&0&0\\ 0&0&-1&0\\ 0&0&0&-1 \end{bmatrix} \qquad G\alpha=\begin{bmatrix}\lambda_1\\ \lambda_2\\ \lambda_1^*\\ \lambda_2^*\\ -\lambda_1\\ -\lambda_2\\ -\lambda_1^*\\ -\lambda_2^*\end{bmatrix} \qquad h=\begin{bmatrix}C\\C\\C\\C\\0\\0\\0\\0\end{bmatrix}$$

▪ Standard form of QP

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

subject to $A\alpha = b$

$G\alpha \le h$

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

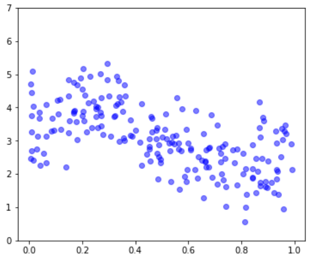
MX-AI

■ Implement a nonlinear SVR from scratch using CVXOPT

```
# [MXML-6-10] 12.cvxopt(svr_nonlinear).py
import numpy as np
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
import matplotlib.pyplot as plt

# Gaussian kernel function.
def kernel(a, b, gamma = 0.1):
    n = a.shape[0]
    m = b.shape[0]
    k = np.array([np.exp(-gamma * np.linalg.norm(a[i] - b[j])**2)
                  for i in range(n)
                  for j in range(m)]).reshape(n, m)
    return k
```

$$k(\vec{x}^{(i)}, \vec{x}^{(j)}) = \exp(-\gamma \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2)$$

```
# Generate sinusoidal data with Gaussian noise added.
def nonlinear_data(n, s):
   rtn_x = []
   rtn_y = []
   for i in range(n):
       x = np.random.random()
       y = np.sin(2.0*np.pi*x)
           + np.random.normal(0.0, s) + 3.0
       rtn_x.append(x)
       rtn_y.append(y)

   return np.array(rtn_x).reshape(-1,1), \
          np.array(rtn_y).reshape(-1,1)

x, y = nonlinear_data(n=200, s=0.7)
```




```
eps = 1.0       # error tolerance range
gamma = 5.0     # gamma for Gaussian kernel
C = 1.0         # regularization constant
n = x.shape[0]

# Construct matrices required for QP in standard form.
K = kernel(x, x, gamma)
P = np.hstack([K, -K])
P = np.vstack([P, -P])
q = np.array([[eps]]) + np.vstack([-y, y])
A = np.array([1.] * n)
A = np.hstack([A, -A])
b = np.zeros((1,1))
G = np.vstack([-np.eye(2*n), np.eye(2*n)])
h = np.hstack([np.zeros(2*n), np.ones(2*n) * C])

P = cvxopt_matrix(P)
q = cvxopt_matrix(q)
A = cvxopt_matrix(A.reshape(1, -1))
b = cvxopt_matrix(b)
G = cvxopt_matrix(G)
h = cvxopt_matrix(h)

# solver parameters
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Perform QP
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
```

Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implement a nonlinear SVR from scratch using CVXOPT

```
# The lambdas, solution to the dual problem
lamb = np.array(sol['x'])

lamb_i = lamb[:n]     # λ
lamb_s = lamb[n:]     # λ*

# calculating the intercept, b
b = []
# 0 < λ < C
idx_i = np.logical_and(lamb_i > 1e-10, lamb_i < C - 1e-10)
if idx_i.shape[0] > 0:
   wx = np.sum((lamb_i-lamb_s) * kernel(x,x[idx_i],gamma),axis=0)
   b.extend((y[idx_i] - wx - eps).flatten())

# 0 < λ* < C
idx_s = np.logical_and(lamb_s > 1e-10, lamb_s < C - 1e-10)
if idx_s.shape[0] > 0:
   wx = np.sum((lamb_i-lamb_s) * kernel(x,x[idx_s],gamma), axis=0)
   b.extend((y[idx_s] - wx + eps).flatten())

if len(b) > 0:
   b = np.mean(b)
   i_sup = np.where(lamb_i > 1e-5)[0]
   s_sup = np.where(lamb_s > 1e-5)[0]

   # Visualize the data, decision function and support vectors
   plt.figure(figsize=(6,5))
   plt.scatter(x, y, c='blue', alpha=0.5)
   plt.scatter(x[i_sup], y[i_sup], c='blue', ec='red', lw=2.0,
               alpha=0.5)
   plt.scatter(x[s_sup], y[s_sup], c='blue', ec='red', lw=2.0,
               alpha=0.5)
```

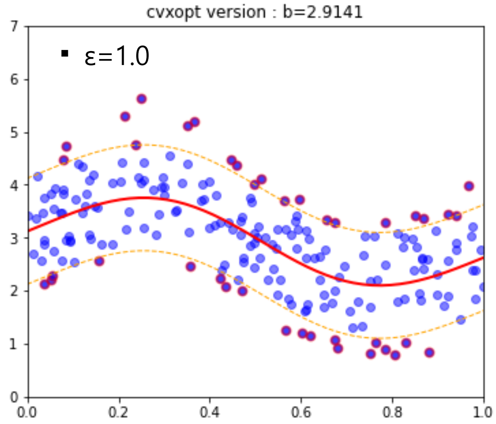
$$\hat{y} = \sum_{i=1}^{n} (\lambda_i - \lambda_i^*) k(\vec{x}^{(i)}, \vec{x}_{test}) + b$$

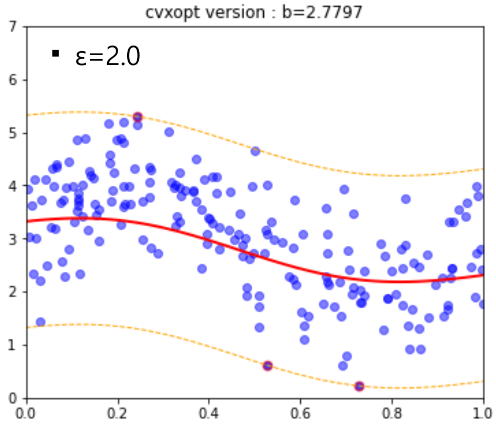
$$\vec{w} \cdot \phi(\vec{x}) = \sum_{i=1}^{n} (\lambda_i - \lambda_i^*) k(\vec{x}^{(i)}, \vec{x})$$

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

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

```
   # decision function
   x_dec = np.linspace(0, 1, 50).reshape(-1, 1)
   wx = np.sum((lamb_i-lamb_s) * kernel(x, x_dec, gamma),\
                axis=0).reshape(-1, 1)
   y_dec = wx + b

   plt.plot(x_dec, y_dec, c='red', lw=2.0)
   plt.plot(x_dec, y_dec + eps, '--', c='orange', lw=1.0)
   plt.plot(x_dec, y_dec - eps, '--', c='orange', lw=1.0)
   plt.xlim(0, 1)
   plt.ylim(0, 7)
   plt.title('cvxopt version : b=' + str(b.round(4)))
   plt.show()
else:
   print('Failed to calculate b.')
```



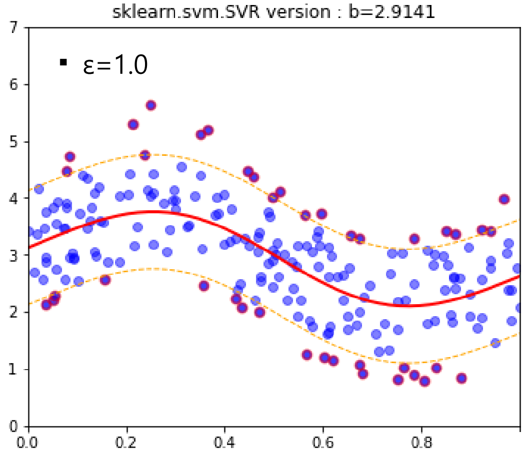


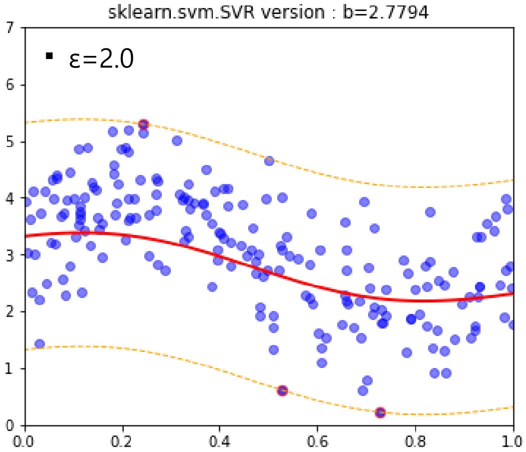
Support Vector Machine (SVM). All rights reserved by www.youtube.com/@meanxai

MX-AI

■ Implement a nonlinear SVR using sklearn's SVR

```
# Compare with the results of sklearn.svm.SVR
from sklearn.svm import SVR
model = SVR(kernel='rbf', gamma=gamma, C=C, epsilon=eps)
model.fit(x, y.flatten())

b1 = model.intercept_
lamb1 = model.dual_coef_
sv_idx = model.support_
sv_x = x[sv_idx]
sv_y = y[sv_idx]

# Visualize the data and decision function
plt.figure(figsize=(5,5))
plt.scatter(x, y, c='blue', alpha=0.5)
plt.scatter(sv_x, sv_y, c='blue', ec='red', lw=2.0, alpha=0.5)
x_dec = np.linspace(0, 1, 50).reshape(-1, 1)
y_dec = model.predict(x_dec)
plt.plot(x_dec, y_dec, c='red', lw=2.0)
plt.plot(x_dec, y_dec + eps, '--', c='orange', lw=1.0)
plt.plot(x_dec, y_dec - eps, '--', c='orange', lw=1.0)
plt.xlim(0, 1)
plt.ylim(0, 7)
plt.title('sklearn.svm.SVR version : b='+str(b1[0].round(4)))
plt.show()
```





* As ε increases or gamma decreases, the regression line becomes flat.