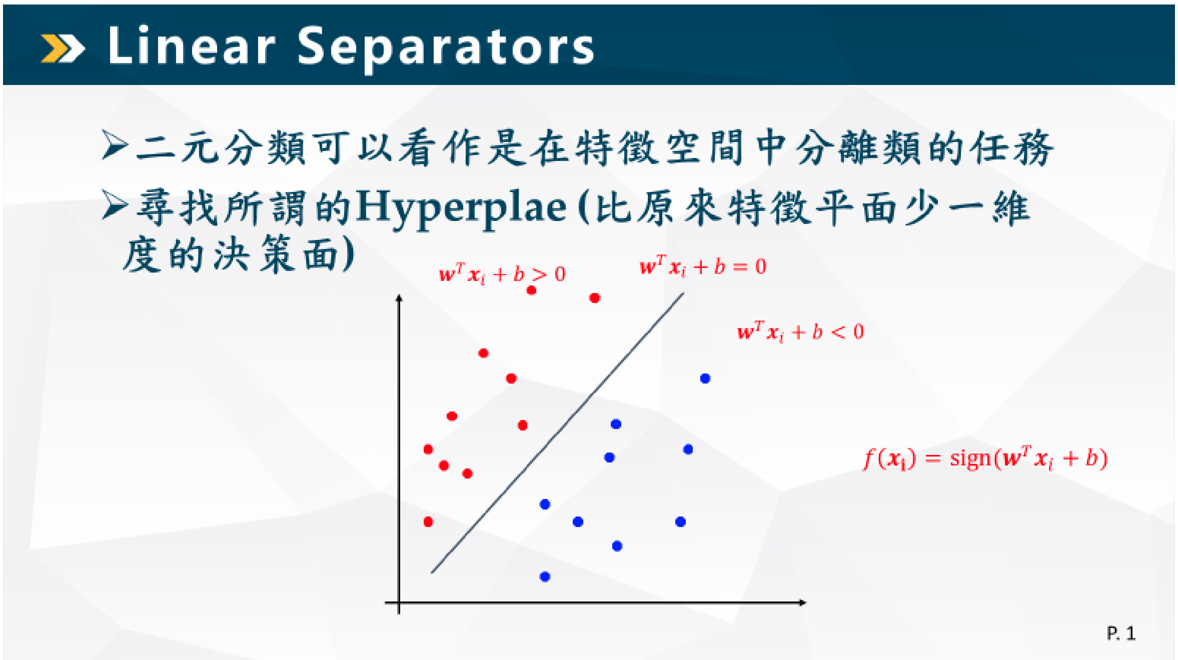
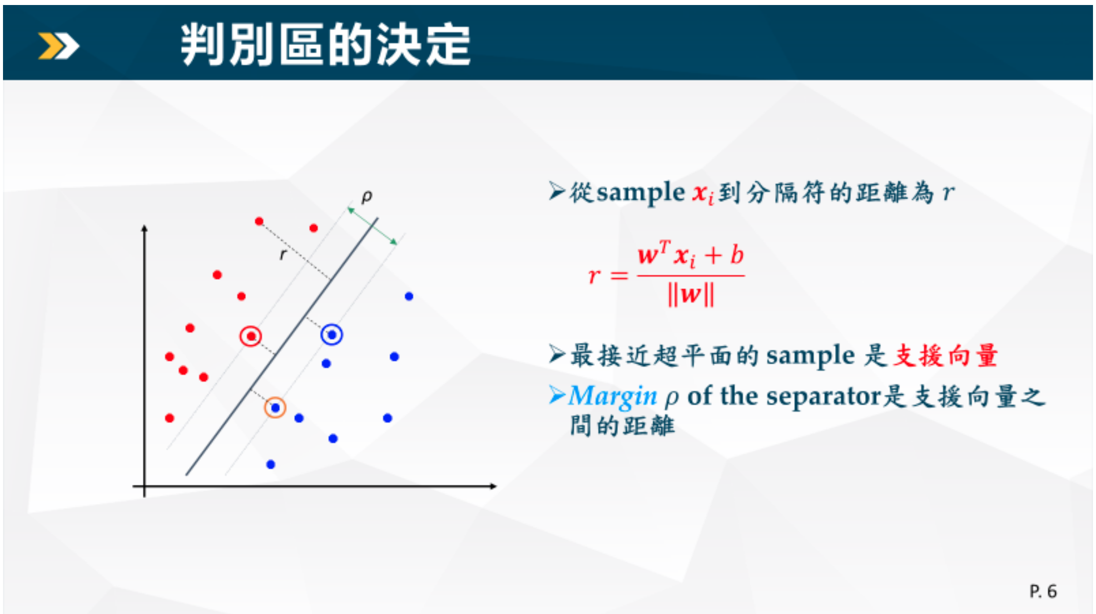
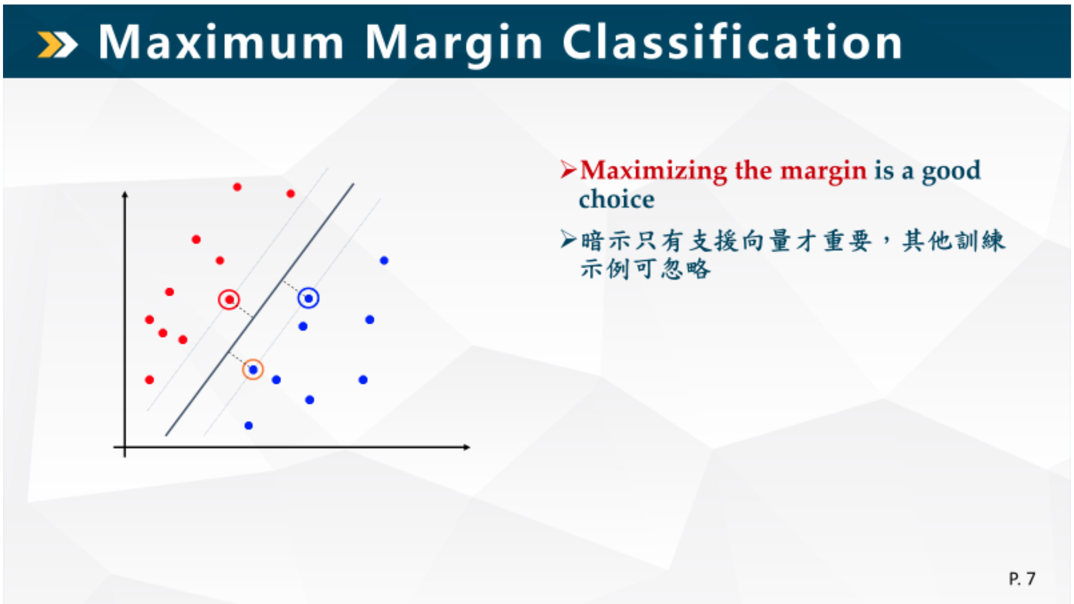
AIoT Lecture 6 Support Vector Machine (SVM)

▼ 1. SVM 介紹

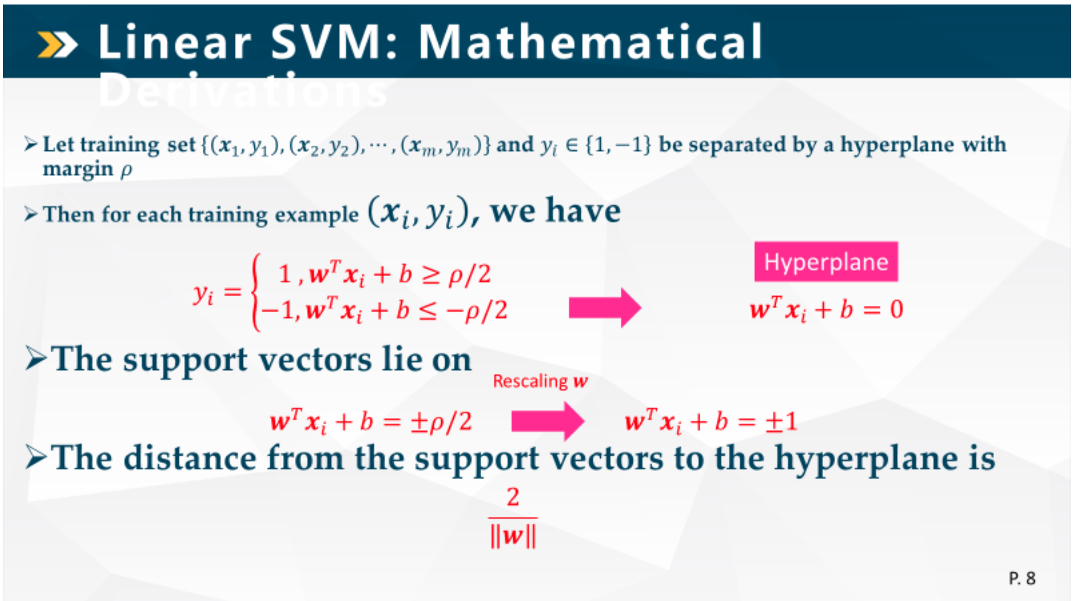


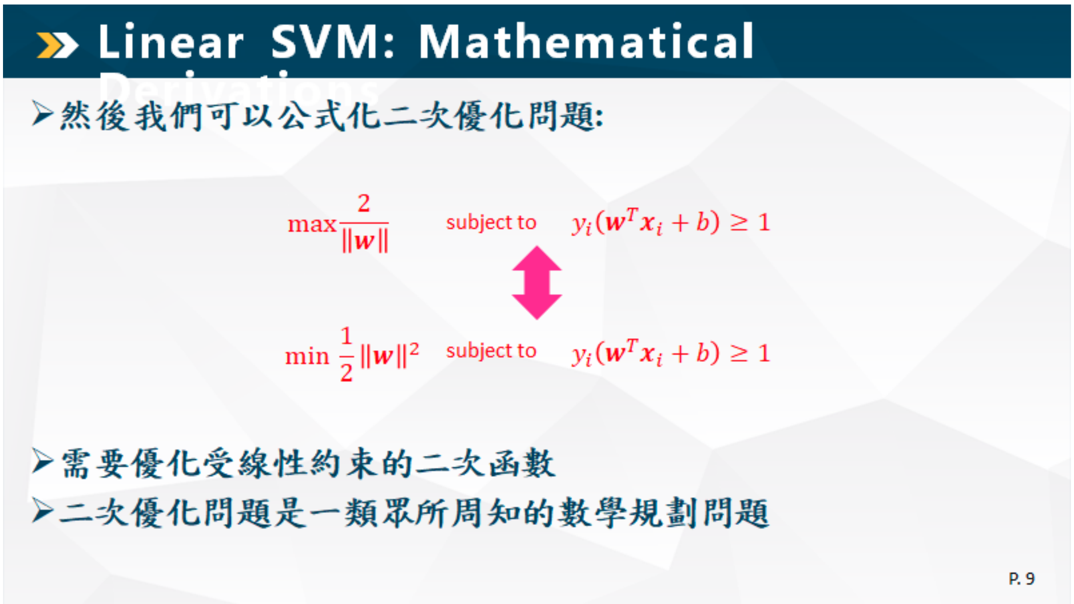




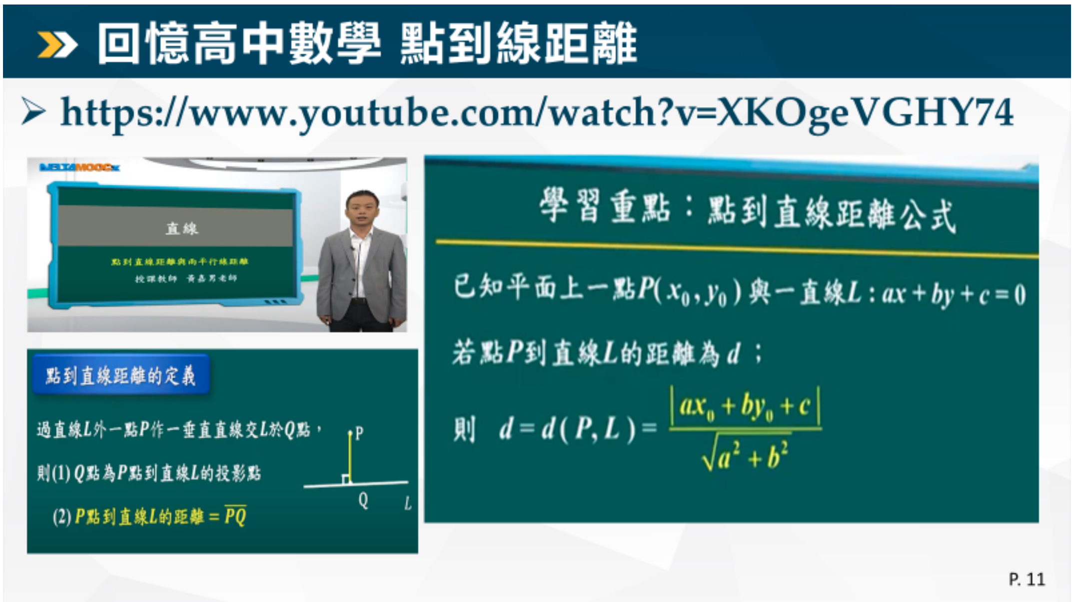
Question: 如何求解

▼ 2. 數學式表達




Linear SVM: Mathematical
然後我們可以公式化二次優化問題:
$\max \frac{2}{\|w\|}$ subject to $y_i(w^T x_i + b) \geq 1$
$\min \frac{1}{2}\|w\|^2$ subject to $y_i(w^T x_i + b) \geq 1$
需要優化受線性約束的二次函數
二次優化問題是一類眾所周知的數學規劃問題
P. 9

▼ 3. 高中線性方程求解


回憶高中數學 點到線距離
https://www.youtube.com/watch?v=XKOgeVGHY74
直線
點到直線距離的定義
過直線L外一點P作一垂直直線交L於Q點，
則(1) Q點為P點到直線L的投影點
(2) P點到直線L的距離 = $\overline{PQ}$
P
Q
L
學習重點：點到直線距離公式
已知平面上一點$P(x_0, y_0)$與一直線$L: ax+by+c=0$
若點P到直線L的距離為d；
則 $d = d(P, L) = \frac{|ax_0+by_0+c|}{\sqrt{a^2+b^2}}$
P. 11

兩平行線距離1

學習重點：兩平行線距離公式

平面坐標系中，

已知兩平行直線$L_1: ax+by+c_1=0$

與$L_2: ax+by+c_2=0$

則兩平行直線的距離為$\dfrac{|c_1-c_2|}{\sqrt{a^2+b^2}}$

P. 12

兩平行線距離2

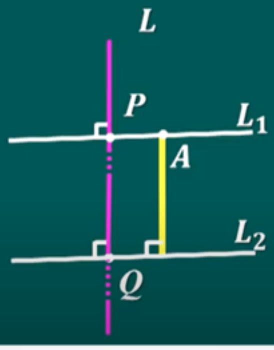
試求兩平行線 $\begin{matrix} L_1: 5x-12y=15 \\ L_2: 5x-12y=2 \end{matrix}$ 的距離？

<解>：

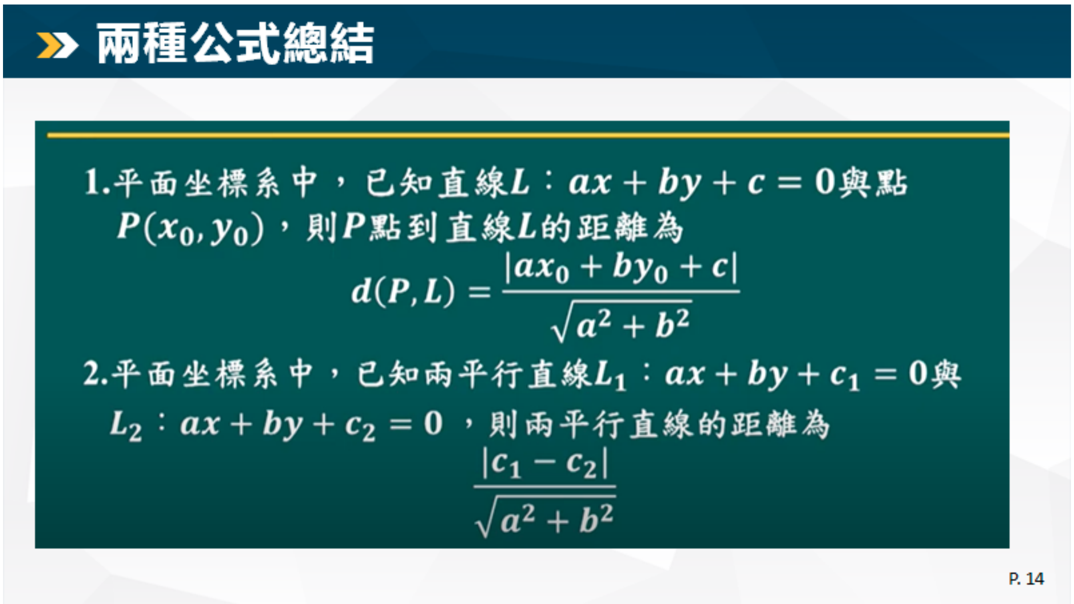
在直線L_1上任取一點$A\,(3,0)$

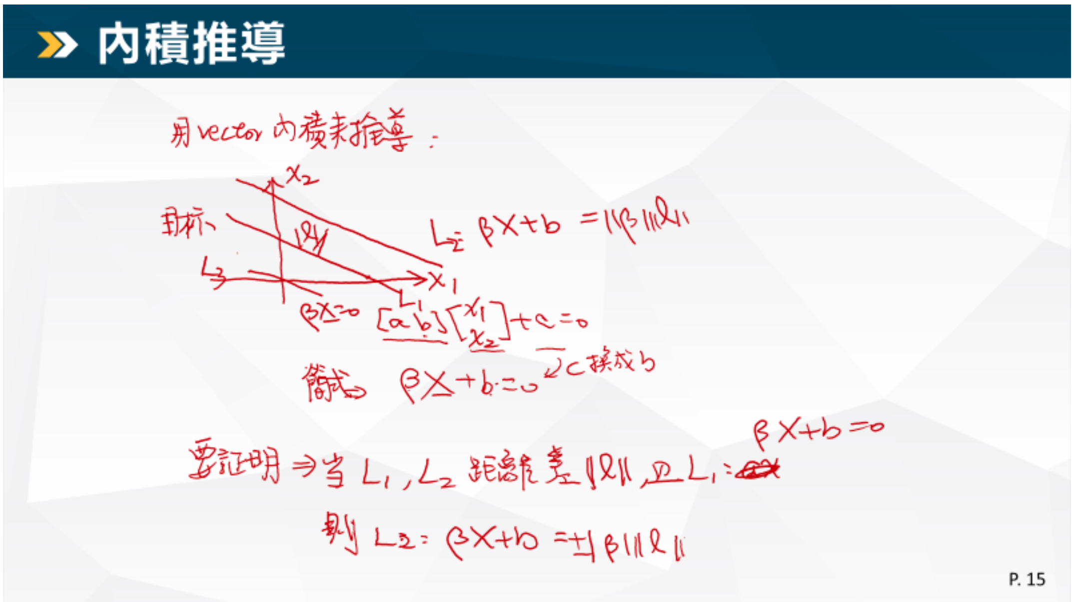
且直線$L_2: 5x-12y-2=0$

則$\mathrm{d}(A,L_2)=\dfrac{|3\times5-12\times0-2|}{\sqrt{5^2+12^2}}=\dfrac{13}{13}=1$



P. 13



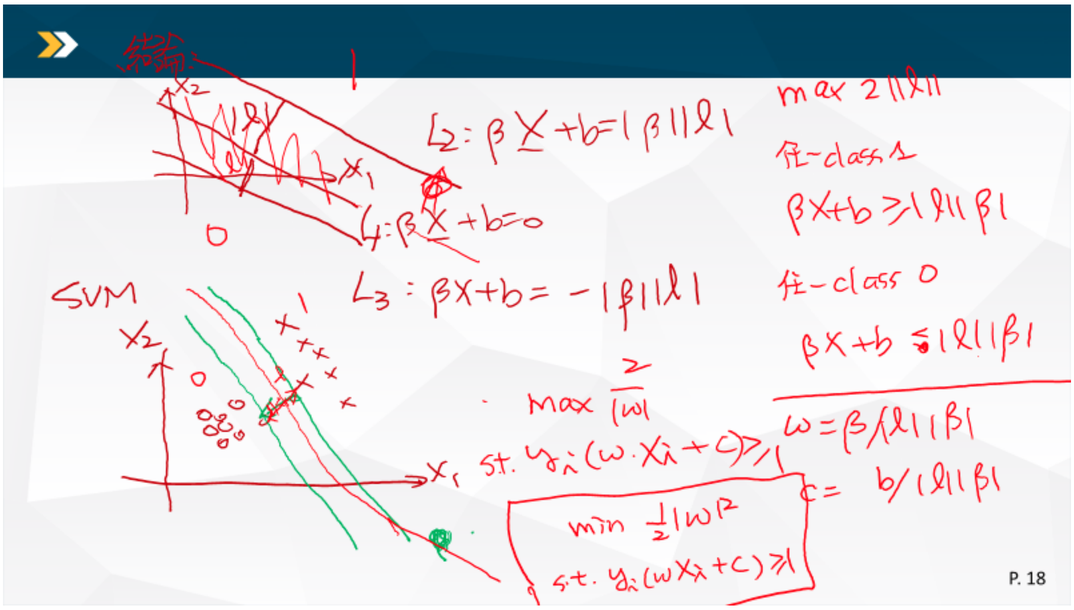


$L+: \beta X+b=|\beta||l|$

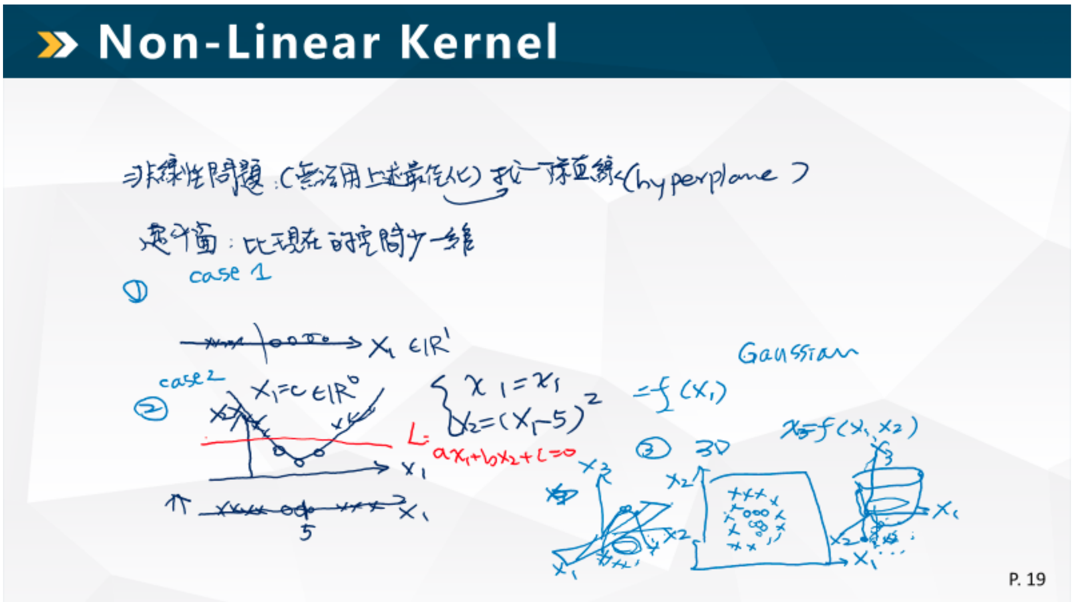
$L1: \beta X+b=0$

$L-: \beta X+b=-|\beta||l|$

公式與兩條線距離對應$|l|=\frac{|c_1-c_2|}{\sqrt{a^2+b^2}}=\frac{||\beta||l|-0|}{|\beta|}$



▼ 4. 非線性Kernel



LinearSVC (線性)
kernel='linear' (線性)
kernel='poly' (非線性)
kernel='rbf' (非線性)

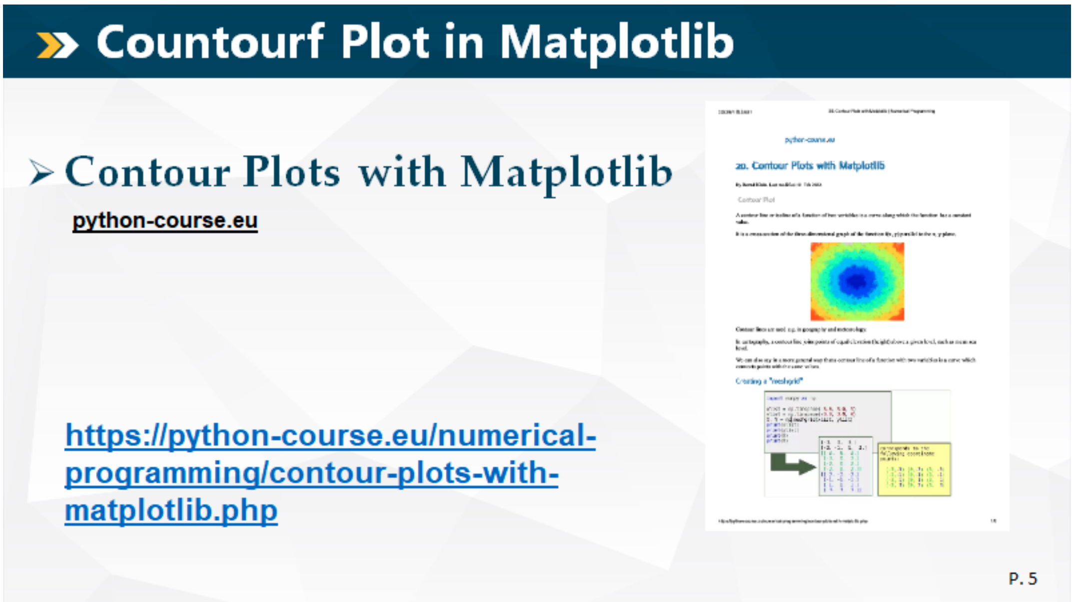
- https://www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/
- https://notes.andywu.tw/2020/白話文講解支持向量機二-非線性svm/

▼ 5. 實作範例觀摩與畫圖工具

- https://medium.com/jameslearningnote/資料分析-機器學習-第3-4講-支援向量機-support-vector-machine-介紹-9c6c6925856b

- https://ithelp.ithome.com.tw/articles/10270447

Countour Plot in Matplotlib



▼ Code example 1 : 線性分兩類的

```
# Topic : ML 分類 2 class (線性)
* follow CRISP-DM


## Step 1: Load data, import library
```

```
import numpy as np


# step 1: Load data, import library

# =====for deep learning ===============

import torch
import torch.nn as nn

#======== for ML ========================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets


# define function
def plotResult(X,y,w1=2, w2=2,b=2):
  plt.scatter(X[y==0,0],X[y==0,1])
  plt.scatter(X[y==1,0],X[y==1,1])
  xm=np.array([X.min(),X.max()])
  ym=(w1*xm+b)/(-w2)
  plt.plot(xm,ym,'r')
  print("w1=",w1,"w2=",w2,"b=",b)
  plt.show()

def getParameters(model):
  w,b =model.parameters()
  w1=w[0][0].item()
  w2=w[0][1].item()
  b=b[0].item()
  return w1,w2,b

# test 1 generate data ==========================
n_samples=200
centers=[[-0.5,0.5],[0.5,-0.5]]
X,y=datasets.make_blobs(n_samples=n_samples,centers=centers,cluster_std=0.4,random_state=3)
#print(X[0],type(X[0]),X[0].shape)
plotResult(X,y)

# test 2 read from 400pts datasets
# data=pd.read_csv("400pts.csv")
# X=data.iloc[:,:-1].values.reshape(-1,2)
# y=data.iloc[:,-1].values
# print(type(X))



print(type(X),X[:5])
print(type(y),y[:5])

# 練習
# plt.scatter(X[y==0,0],X[y==0,1])
# # plt.show()
# plt.scatter(X[y==1,0],X[y==1,1])
# plt.show()


## Step 2: Preprocessing

#step 2 :  Preprocessing for pytorch
tensor_X=torch.FloatTensor(X)
tensor_y=torch.FloatTensor(y.reshape(200,1))
print(tensor_X.size())

# Step 3: Build Model

# step 3:
from sklearn.linear_model import LogisticRegression as LR
model=LR()
model.fit(X,y)
print(model.coef_,model.intercept_)
[[w1,w2]]=model.coef_
[b]=model.intercept_
print('Logistic regression')
plotResult(X,y,w1=w1, w2=w2,b=b)


# step 3:
# from sklearn.svm import SVC
# model2=SVC(kernel="linear")
# print(model2.get_params())
# model2.fit(X,y)
# print(model2.coef_,model2.intercept_)
# [[w1,w2]]=model2.coef_
# [b]=model2.intercept_
```

```
# print('SVC')
# plotResult(X,y,w1=w1, w2=w2,b=b)



<img src= "https://lh6.googleusercontent.com/pgnVbPBuXCiR5Ri2t0R9xIl4rpuTXWWZ5euvGhegi8DH8H0Lbcu19DwX86f4C5OhexXsOI9V6iIG9mUcAGBw1


#step 3: build model

class MyLogRegNN(nn.Module):
  def __init__(self,inSize,outSize):
    super().__init__()
    self.linear=nn.Linear(inSize,outSize)
  def forward(self,x):
      x=self.linear(x)
      y_hat=torch.sigmoid(x)
      return y_hat
  def predict(self,x):
      pred=self.forward(x)
      if pred >=0.5:
        return 1
      return 0


model=MyLogRegNN(2,1)
print(model)


a=[1,2,3,4,5,6]
a_it=iter(a)
for i in range(6):
  print(next(a_it))

#step 4 training
criterion=nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(),lr=0.8)
epochs=100
losses=[]
for e in range(epochs):
  #compute loss
  y_hat=model(tensor_X)
  loss_=criterion(y_hat,tensor_y).detach().numpy()
  loss=criterion(y_hat,tensor_y)

  losses.append(loss_)

  # 3 steps for Gradient update
  optimizer.zero_grad()
  loss.backward()
  optimizer.step()


plt.plot(range(epochs),losses)



# plot results

w1,w2,b =getParameters(model)
plotResult(X,y,w1,w2,b)
```

▼ Code example 2: 非線性分兩類

```
Topic: Classification (ML+DL)
Step 1: load data, import library
[ ]
# step 1: Load data, import library
# import torch
# import torch.nn as nn

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets

# define function

# def getParameters(model):
#   w,b =model.parameters()
#   w1=w[0][0].item()
```

```
#   w2=w[0][1].item()
#   b=b[0].item()
#   return w1,w2,b

# generate data =================
n_samples=500
X,y=datasets.make_circles(n_samples=n_samples,noise=0.2,random_state=3,factor=0.2)
#print(X[0],type(X[0]),X[0].shape)


[ ]


def plotBoundary(X,y,model):
  row_condition= (y==0)

  plt.scatter(X[row_condition,0],X[y==0,1])
  plt.scatter(X[y==1,0],X[y==1,1])

  x_span=np.linspace(min(X[:,0])-0.25,max(X[:,0])+0.25)
  y_span=np.linspace(min(X[:,1])-0.25,max(X[:,1])+0.25)
  xx,yy=np.meshgrid(x_span,y_span)
  print(x_span.shape,xx.shape)
  grid=np.c_[xx.ravel(),yy.ravel()]

  z=model.predict_proba(grid)
  print(type(z),z.shape)
  plt.contour(xx,yy,z)
  plt.show()

[ ]
plt.scatter(X[y==0,0],X[y==0,1])
plt.scatter(X[y==1,0],X[y==1,1])
plt.show()


Step 2 : Preprocessing
[ ]
#step 2 : Preprocessing for ML
print(type(X),X.shape)
print(type(y),y.shape)
print(y[:5])

#step 2 : Preprocessing for DL
# tensor_X=torch.FloatTensor(X)
# tensor_y=torch.FloatTensor(y.reshape(n_samples,1))
# print(tensor_X.size())
# print(tensor_y.size())
<class 'numpy.ndarray'> (500, 2)
<class 'numpy.ndarray'> (500,)
[0 1 0 0 0]

Step 3: Build Model
參考blog

[ ]
def plotBoundary(X,y,model):
  row_condition= (y==0)

  plt.scatter(X[row_condition,0],X[y==0,1])
  plt.scatter(X[y==1,0],X[y==1,1])

  x_span=np.linspace(min(X[:,0])-0.25,max(X[:,0])+0.25)
  y_span=np.linspace(min(X[:,1])-0.25,max(X[:,1])+0.25)
  xx,yy=np.meshgrid(x_span,y_span)
  print(x_span.shape,xx.shape)
  grid=np.c_[xx.flatten(),yy.flatten()]

  z=model.predict_proba(grid)
  z0=z[:,0].reshape(xx.shape)
  print(type(z0),z0.shape)
  plt.contour(xx,yy,z0,alpha=0.5)
  plt.show()
[ ]
# step 3: Build model for ML
# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
model = SVC(kernel ='rbf',random_state = 0,probability=True)

model.fit(X, y)

y_hat=model.predict_proba(X)
print(y_hat[:5])

plotBoundary(X,y,model)

# plt.scatter(X[y==0,0],X[y==0,1])
```

```
# plt.scatter(X[y==1,0],X[y==1,1])

# x_span=np.linspace(min(X[:,0])-0.25,max(X[:,0])+0.25)
# y_span=np.linspace(min(X[:,1])-0.25,max(X[:,1])+0.25)
# xx,yy=np.meshgrid(x_span,y_span)
# print(x_span.shape,'xx.shape=',xx.shape)
# grid=np.c_[xx.ravel(),yy.ravel()]
# print(grid.shape)
# z=model.predict_proba(grid)
# print('z=',z)
# print(type(z),z.shape)
# plt.contour(xx,yy,z)
# # plt.show()


[ ]
#step 3: build model for DL

# class MyDNN(nn.Module):
#   def __init__(self,inSize,h1Size,h2Size,outSize):
#     super().__init__()
#     self.linear=nn.Linear(inSize,h1Size)
#     self.linear2=nn.Linear(h1Size,h2Size)
#     self.linear3=nn.Linear(h2Size,outSize)
#   def forward(self,x):
#       x=self.linear(x)
#       x=torch.sigmoid(x)
#       x=self.linear2(x)
#       x=torch.sigmoid(x)
#       x=self.linear3(x)
#       x=torch.sigmoid(x)
#       return x
#   def predict(self,x):
#       pred=self.forward(x)
#       if pred >=0.5:
#         return 1
#       return 0


# model=MyDNN(2,10,4,1)
# print(model)

[ ]

[ ]
# #step 4 training
# criterion=nn.BCELoss()
# optimizer = torch.optim.Adam(model.parameters(),lr=0.01)
# epochs=5000
# losses=[]
# for e in range(epochs):
#   #compute loss
#   if e%1000 ==0: print("*",end="")
#   y_hat=model(tensor_X)
#   loss=criterion(y_hat,tensor_y)
#   losses.append(loss)

#   # 3 steps for Gradient update
#   optimizer.zero_grad()
#   loss.backward()
#   optimizer.step()
# plt.plot(range(epochs),losses)


[ ]
# plot results
plotBoundary(X,y,model)

[ ]
```