# 訓練模型前置動作

1. **Split data (將資料集依比例自動分配成可以提供給模型訓練和測試的樣本)**

from sklearn.model\_selection import train\_test\_split

x =…

y =… >>>作為答案集

x\_train(訓練集), x\_test(測試集), y\_train(訓練答案集), y\_test(測試答案集) =

train\_test\_split(x, y, test\_size=0.3, random\_state=1)

>>>代表分配訓練:測試 = 7:3

>>>不固定random\_state數字的話,這四集每次print出來的結果都會不一樣

(隨機分配)

# Preprocessing Data(將資料標準化)

from sklearn import preprocessing

x\_train = … #訓練集

scaler = preprocessing.StandardScaler().fit(x\_train) #建立模型

x\_train = scaler.transform(x\_train) #訓練模型

(((以上兩行可以直接簡化成 :

x\_train = preprocessing.StandardScaler().fit\_transform(x\_train) )))

x\_test = … #測試集

x\_test = scaler.transform(x\_test) #測試模型

# 分類Classification(監督式學習)

# 線性迴歸(Linear Regression)

* **實作**

from sklearn import linear\_model

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_train(訓練集), df\_train\_y(答案集), test\_size=0.2)

model = linear\_model.LinearRegression()

model.fit(x\_train, y\_train) #訓練模型

y\_pred = model.predict(x\_test) #預測y\_test

* **其他計算**
* model.coef\_ #斜率
* model.intercept\_ #截距

# 羅吉斯迴歸模型（Logistic Regression）

* 把y壓縮為0或1,解決二元分類問題或機率問題
* 實作

from sklearn.linear\_model import LogisticRegression

model=LogisticRegression(max\_iter = 2000)

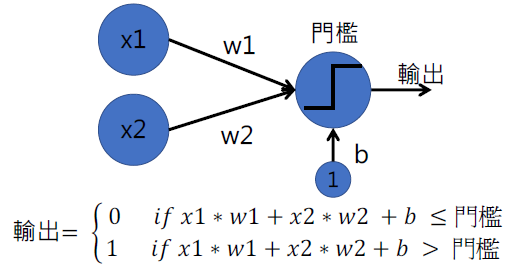
>>>訓練次數(越高越容易overfitting,預設100)

model.fit(x\_train,y\_train)

y\_pred = model.predict(x\_test) #預測y\_test

* model.predict\_proba(x\_test [:2]) #預測前兩筆或然率(得到y各種結果的機率)

# 類神經網路



* **感知器Perceptron（PLA）**就像是在模擬神經元,透過**激活函數(Activation Function)**計算複雜問題並進行分類
* **激活函數** : 通常是一個非線性的函數,才可以將網路複雜化
* 實作

from sklearn.linear\_model import Perceptron

model = Perceptron().fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

* model.coef\_
* model.intercept\_

# 決策樹(ID3)

* 原理 : 希望切分後的資料相似程度很高(亂度減低) → 計算訊息增益(Information Gain) = Entropy(before splitting) -E[Entropy(after splitting)]
* 實作

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(min\_samples\_leaf=3)

>>>最終的葉子(節點)上至少要有多少樣本

model.fit(x\_train, y\_train)

* 畫圖

from sklearn import tree

tree.plot\_tree(model)

* **隨機森林模型 :** 為了解決決策樹Overfitting的缺點而生。

用一樣的決策樹,隨機對樣本採樣產生多棵決策樹,再丟入測試集並計算哪顆樹效果最好

* 實作

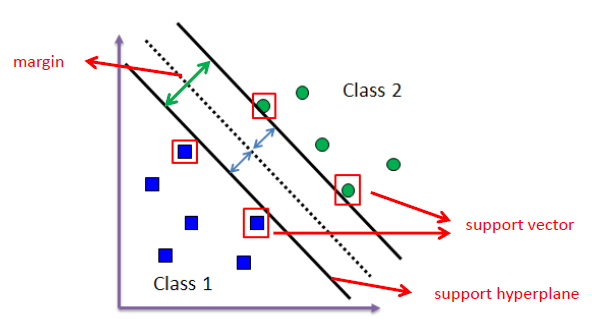
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(max\_depth=6, n\_estimators=10)

>>>樹能生長到幾層/產生幾顆決策樹

# 支持向量機SVM

* 目的 : 找到可最大程度分離數據的切線



>>>margin越大越好,至少要大於等於1

>>>通常定義class 1和class 2的y值為**1和-1**

* 對於線性不可分的問題,可透過Kernel參數把資料映射到高維度



* 實作

from sklearn.svm import SVC

model = SVC(kernel='rbf') >>>預設(最常用)

model.fit(x\_train, y\_train)

# 樸素貝葉斯(Naive Bayes)

* 利用貝氏定理做分類
* 實作

from sklearn.naive\_bayes import GaussianNB(針對連續型資料) 或 MultinomialNB(針對離散型資料,避免0的出現)

model = GaussianNB() >>>資料是連續型的話

model.fit(x\_train, y\_train)

# 集成學習 - 將多種分類器結合在模型中

# Bagging(Bootstrap aggregating)

* 同一種分類器但對樣本採樣多次以訓練模型
* Ex : **隨機森林**

# Boosting

* 前一個分類器分錯的樣本會得到加權，下一個分類器再對加權後的全體樣本進行訓練，最後把所有分類器結合起來
* Ex: **Adaboost**
* 實作

from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(n\_estimators=100, random\_state=0)

#預設50棵樹

# Blending(最常見)

* 用同樣的樣本訓練好各分類器後,依表現分配權重並加總在模型中
* 實作

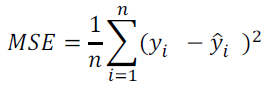
訓練好各分類器後…..

MSE加總 = 1/第1個分類器的MSE + 1/第2個分類器的MSE + 1/第3個分類器的MSE >>>讓誤差越大的權重越小

Blending預測值 = 第1個分類器的預測值\*((1/第1個分類器的MSE)/ MSE加總) + 第2個分類器的預測值\*((1/第2個分類器的MSE)/ MSE加總) + 第3個分類器的預測值\*((1/第3個分類器的MSE)/ MSE加總)

# 評估模型

1. **均方誤差(MSE)**

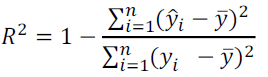


* 和預測值的差異,越接近0越好
* from sklearn.metrics import mean\_squared\_error

mean\_squared\_error(y\_test, y\_pred,squared=False)

>>>通常平方完會開平方(RMSE)

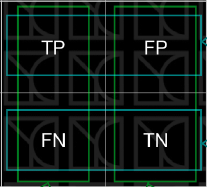
1. **R squared**



* 越接近1越好
* from sklearn.metrics import r2\_score

r2\_score(y\_test, y\_pred)

1. **混淆矩陣(Confusion matirx)**



* from sklearn.metrics import confusion\_matrix

confusion\_matrix([答案矩陣], [預測矩陣])

>>>長得像這樣的矩陣

* **Accuracy :** TP+TN/ALL

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred, normalize=True)

>>>內建True : get accuracy score/False : get number of correct samples

* **Precision :** TP/TP+FP

from sklearn.metrics import precision\_score

precision\_score(y\_test, y\_pred, zero\_division=1)

>>>結果為0時⇒自動變為1

* **Recall :** TP/TP+FN

from sklearn.metrics import recall\_score

recall\_score(y\_test, y\_pred)

# 分群clustering(非監督式學習)

# 階層式分群（Hierarchical Clustering）



* 依距離做分群
* from sklearn.cluster import AgglomerativeClustering #由下而上

clustering = AgglomerativeClustering(n\_clusters = 要分幾群, linkage = "距離的算法(如下)", distance\_threshold = 限制群內的最大距離).fit(X)

# ward = 最小化群內各點到群中心的距離平方和

# complete = 兩個群中最遠的點

# average = 兩個群中各點與各點間距離總和的平均

# single = 兩個群中最近的點

* clustering.predict(輸入X) #預測分在哪一群
* clustering.labels\_ >>>把X分成兩群的所屬群標籤

# 分裂式分群（K-means）



* 依要分幾群隨機設立群中心，看資料離哪個群中心最近就是哪群，然後不斷迭代
* K-Means 隨機初始化陷阱：起始點會影響到最終的分群結果→ local optimum
* from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=要分成幾個群, random\_state=0).fit(X)

* kmeans.predict(輸入X) #預測分在哪一群
* kmeans.labels\_ #各個X所屬群標籤
* kmeans.cluster\_centers\_ #看群中心是哪兩個X

# 密度式分群（DBSCAN）



* 依半徑做分群,無法設定要分幾群
* from sklearn.cluster import DBSCAN

clustering = DBSCAN(eps=半徑, min\_samples=一個圈內至少要有幾個資料點).fit(X)

* clustering.predict(輸入X) #預測分在哪一群
* clustering.labels\_ #各個X所屬群標籤

>>>array([0, 1, -1]) #-1代表noise point（離群值）