|  |  |  |
| --- | --- | --- |
| No. | Code. | Des. |
| 1 | #import libraries  import random  import numpy as np  import matplotlib.pyplot as plt  from matplotlib.colors import ListedColormap  import seaborn as sns  import pandas as pd  from sklearn.model\_selection import train\_test\_split  import sklearn.metrics as metric  import plotly.express as px | เป็นนำ library ต่างที่จะใช้งาน เช่น pandas ใช้สำหรับ dataframe เป็นต้น |
| 2 | #Load Data  from sklearn.datasets import load\_wine | Load ข้อมูลเรื่อง น้ำไวน์องุ่น |
| 3 | def generate\_data(size):  x = []  y = []  target = []  for i in range(size):  x.append(np.round(random.uniform(0,2.5),1))  y.append(np.round(random.uniform(0,20),1))  target.append(0)    x.append(np.round(random.uniform(1,5),2))  y.append(np.round(random.uniform(20,25),2))  target.append(1)    x.append(np.round(random.uniform(3,5),2))  y.append(np.round(random.uniform(5,25),2))  target.append(1)  df\_x = pd.DataFrame(x)  df\_y = pd.DataFrame(y)  df\_target = pd.DataFrame(target)  df = pd.concat([df\_x,df\_y],ignore\_index = True ,axis = 1)  df = pd.concat([df,df\_target],ignore\_index= True,axis=1)  df.columns = ["X","Y","Target"]  return df | สร้าง function ที่ทำงานเป็นการสร้างชุดข้อมูลจากการสุ่มตัวเลข |
| 4 | wine = load\_wine()  print(wine.keys()) | สร้างตัวแปรมารองรับข้อมูลที่โหลดมา  และแสดงหัวข้อของข้อมูล |
| 5 | wine.feature\_names | แสดงชื่อ columns ของ feature หรือ ตัวแปร |
| 6 | wine\_data = wine.data  wine\_target = wine.target | สร้างตัวแปรไว้เก็บข้อมูลจริงๆที่ตัวนั้นโหลดที่มีหลากหลาย |
| 7 | wx,wx\_test,wy,wy\_test = train\_test\_split(wine\_data,wine\_target,test\_size =0.2,random\_state=42) | สร้างตัวแปรไว้สำหรับ train test โดยมีการกำหนด seem random = 42 |
| 8 | wx\_train,wx\_val,wy\_train,wy\_val = train\_test\_split(wx,wy,test\_size=0.2,random\_state=42) | สร้างตัวแปรไว้สำหรับ train test โดยมีการกำหนด seem random = 42 |
| 9 | wx\_train | แสดงข้อมูล |
| 10 | wy\_train | แสดงข้อมูล |
| 11 | size = 100  data = generate\_data(size)  features = data[["X","Y"]]  label = data[["Target"]] | สร้างตัวแปร size =100 และสร้างตัวแปรdata ที่สร้างจากการสุ่มตัวเลข  และแบ่งชุดข้อมูลเป็น feature กับ ตัว labels |
| 12 | test\_size = int(np.round(size\*0.2,0))  print(test\_size) | สร้างตัวแปร test\_size = 20 |
| 13 | x\_train = features[:-test\_size].values  y\_train = label[:-test\_size].values  x\_test = features[-test\_size:].values  y\_test = label[-test\_size:].values | แบ่งชุดข้อมูลเป็น train test |
| 14 | fig,ax = plt.subplots(figsize = (10,10))  ax.spines["top"].set\_visible(False)  ax.spines["left"].set\_visible(False)  ax.spines["right"].set\_visible(False)  ax.scatter(features[:-test\_size]["X"],features[:-test\_size]["Y"],color = "#8C7298")  ax.grid(color = "grey",linestyle="-",linewidth=0.25) | Plot กราฟ เป็นจุดสีม่วง |
| 15 | #Implement and train liner models  #SVM  from sklearn.svm import SVC | เริ่มสร้าง model อันแรกคือ svm |
| 16 | svm = SVC(kernel = 'rbf',random\_state=0) | สร้างตัว model โดยกำหนดพารามิเตอร์ต่างๆ |
| 17 | svm.fit(x\_train,y\_train) | ทำนายผล |
| 18 | fig,ax = plt.subplots(figsize=(10,10))  ax.spines["top"].set\_visible(False)  ax.spines["left"].set\_visible(False)  ax.spines["right"].set\_visible(False)  xx = np.linspace(-1,max(features["X"])+1,len(x\_train))  yy = np.linspace(0,max(features["Y"])+1,len(y\_train))  yy,xx = np.meshgrid(yy,xx)  xy = np.vstack([xx.ravel(),yy.ravel()]).T  train\_size = len(features[:-test\_size]["X"])  colors = y\_train  colors = np.where(colors == 1,"#8C7298","#4786D1")  ax.scatter(features[:-test\_size]["X"],features[:-test\_size]["Y"],c = np.concatenate(colors,axis = 0))  z = svm.decision\_function(xy).reshape(xx.shape)  ax.contour(xx,yy,z,colors="k",levels = [-1,0,1] ,alpha = 0.5,linestyles = ["--","-","--"])  ax.scatter(svm.support\_vectors\_[:,0],svm.support\_vectors\_[:,1],s = 100,linewidth =1,  facecolors='none',edgecolors='k')  ax.grid(color ='grey',linestyle="-",linewidth=0.25) | Plot กราฟ ที่ SVM ทำนายในการแบ่งแยกข้อมูล |
| 19 | #K-NN  from sklearn.neighbors import KNeighborsClassifier | Model K-NN |
| 20 | knn = KNeighborsClassifier(n\_neighbors = 3) | สร้าง model |
| 21 | x = wx\_train[:,:2]  y = wy\_train | แบ่งข้อมูล |
| 22 | knn.fit(x,y) | ทำนายหรือแบ่งแยกข้อมูล |
| 23 | cmap\_light = ListedColormap(['#FFAAAA','#AAFFAA','#AAAAFF'])  cmap\_bold = ListedColormap(['#FF0000','#00FF00','#0000FF']) | กำหนดค่าสี |
| 24 | x\_min,x\_max = x[:,0].min() -1 ,x[:,0].max() +1  y\_min,y\_max = x[:,1].min() -1,x[:,1].max() +1  xx,yy = np.meshgrid(np.arange(x\_min,x\_max,0.02),np.arange(y\_min,y\_max,0.02))  z = knn.predict(np.c\_[xx.ravel(),yy.ravel()])  z = z.reshape(xx.shape)  plt.pcolormesh(xx,yy,z,cmap = cmap\_light)  plt.scatter(x[:,0],x[:,1],c = y,cmap=cmap\_bold)  plt.xlim(xx.min(),xx.max())  plt.ylim(yy.min(),yy.max()) | สร้างตัวแปรจากการแบ่งข้อมูลบางส่วนของข้อมูลอันที่แล้ว  และ plot จุดเป็นสีและทำคล้าย voronoid |
| 25 | #Decosopm Tree  from sklearn.tree import DecisionTreeClassifier  from sklearn import tree | Import model decision tree |
| 26 | dtc = DecisionTreeClassifier() | สร้าง model |
| 27 | dtc.fit(wx\_train,wy\_train) | ทำนาย model |
| 28 | y\_im = wine.target\_names.tolist() | เปลี่ยนตัวแปรให้เป็น list(เปลี่ยนชนิดของขอ้มูลเฉยเพราะต้องนำไปต่อไป) |
| 29 | plt.figure(figsize=(10,10))  tree.plot\_tree(dtc,feature\_names = wine.feature\_names,  class\_names = y\_im,filled = True) | Plot กราฟ tree ในการตัดสินใจว่าเลือกแบบไหน |
| 30 | #Random Forest  from sklearn .ensemble import RandomForestClassifier | นำ library random forest |
| 31 | rf = RandomForestClassifier(n\_estimators = 6) | สร้าง model |
| 32 | rf.fit(wx\_train,wy\_train) | ทำนาย model |
| 33 | fn = wine.feature\_names  cn = wine.target\_names.tolist()  fig,ax = plt.subplots(2,3,figsize=(50,20))  ax = ax.flatten()  for index in range(6):  tree.plot\_tree(rf.estimators\_[index],feature\_names=fn,  class\_names =cn,filled =True,ax=ax[index])  ax[index].set\_title("Estimator:"+str(index+1)) | Plot กราฟป่า(ต้นไม้หลายต้น) |
| 34 | #Multi-layer Perceptron  class MLP():  print('Running...')  def \_\_init\_\_(self,design\_matrix,Y,iterations=1000000,lr=1e-1,  input\_layer=2,hidden\_layer =3,output\_layer=1):  self.design\_matrix = design\_matrix #design matrix attibute  self.iterations = iterations #ITERATIONS ATTIBUTE  self.lr = lr #learning rate attibute  self.input\_layer = input\_layer #input layer attibute  self.hidden\_layer = hidden\_layer  self.output\_layer = output\_layer #output layer attibute  self.weight\_matrix\_1 = np.random.randn(self.input\_layer,self.hidden\_layer)  #weight attribute connecting to the hidden layer  self.weight\_matrix\_2 = np.random.randn(self.hidden\_layer,self.output\_layer)#to the output layer  self.cost = [] #cost list attribute  self.p\_hats = [] #predictions list attirbute  def sigmoid(self,x):  #sigmoid functyion used at ther hidden layer and output layer  return 1/(1+np.exp(-x))  def sigmoid\_derivative(self,x):  #sigmoid derivative used fro backpropgation  return self.sigmoid(x) \* (1 - self.sigmoid(x))  def forward\_propagation(self):  #define function to feedforward the network  z = np.dot(self.design\_matrix,self.weight\_matrix\_1) #liner tranformation to the hidden layer  activation\_func = self.sigmoid(z) #hidden layer activation function  zh = np.dot(activation\_func,self.weight\_matrix\_2)  p\_hat = self.sigmoid(zh)  return z,activation\_func,zh,p\_hat  def BCECost(self,y,p\_hat):  #bianry cross entropy cost function  bce\_cost = -(np.sum(y\*np.log(p\_hat)+(1-y)\*np.log(1-p\_hat))/len(y))  return bce\_cost  def backword\_prop(self,z\_1,activation\_func,z\_2,p\_hat):  del\_2\_1 = p\_hat -Y  partial\_deriv\_2 = np.dot(activation\_func.T,del\_2\_1)  del\_1\_1 = del\_2\_1  del\_1\_2 = np.multiply(del\_1\_1,self.weight\_matrix\_2.T)  del\_1\_3 = np.multiply(del\_1\_2,self.sigmoid\_derivative(z\_1))  partial\_deriv\_1=np.dot(self.design\_matrix.T,del\_1\_3)  return partial\_deriv\_2,partial\_deriv\_1  def train(self):  for i in range(self.iterations):  z\_1,activation\_func,z\_2,p\_hat = self.forward\_propagation()  partial\_deriv\_2,partial\_deriv\_1 = self.backword\_prop(z\_1,activation\_func,z\_2,p\_hat)  self.weight\_matrix\_1 = self.weight\_matrix\_1 - self.lr \* partial\_deriv\_1  self.weight\_matrix\_2 = self.weight\_matrix\_2 - self.lr \* partial\_deriv\_2  self.cost.append(self.BCECost(Y,p\_hat))  self.p\_hats.append(p\_hat)  print('Training Complete')  print('------------------------------------------------------------------') | สร้าง class multi-layer perceptron  ตรง init ต้องมี underscore 2 อันเท่านั้น  รายละเอียดข้างในมีฟังก์ชัน sigmoid sigmoid diff  Forward\_propagation BCECost= การวัดค่า error และต่อมาคือ backword\_propagation และสุดท้ายคือ train |
| 35 | X = np.array([[1,0],[0,1],[0,0],[1,1]])  Y = np.array([[1],[1],[0],[0]]) | สร้างตัวแปรที่เก็บค่าของ np array ไว้ |
| 36 | mlp = MLP(X,Y)  mlp.train() | สร้าง model and predict it |
| 37 | print(f'\n The MLP predictions for each training example,based on{mlp.iterations} iteractions are:\n\n{np.round(mlp.p\_hats[-1],2)}')  print('\n---------------------------------------------------------------------------')  print(f'\n The ground truth Y lables are are:\n\n{Y}') | แสดงค่าที่ predict ที่รันผ่าน แสนครั้งรอบวนและแสดงค่าจริงที่ labels ไว้หรือเรียกอย่างว่า ground truth |
| 38 | #Neural Network  import tensorflow as tf  from tensorflow import keras | Import library |
| 39 | model = keras.models.Sequential()  activation = 'relu'  model.add(keras.layers.Flatten(input\_shape=[13]))  model.add(keras.layers.Dense(400,activation=activation))  model.add(keras.layers.Dense(200,activation=activation))  model.add(keras.layers.Dense(100,activation=activation))  model.add(keras.layers.Dense(50,activation=activation))  model.add(keras.layers.Dense(3,activation="softmax")) | สร้าง model  กำหนด activation function คือ relu  กำหนด unit ต่างๆในเครือข่ายสมอง |
| 40 | model.summary() | ดูผลต่างๆโดยรวม |
| 41 | model.compile(loss="sparse\_categorical\_crossentropy",optimizer = "adam",metrics = ["accuracy"]) | สร้าง model |
| 42 | history = model.fit(wx\_train,wy\_train,validation\_data = (wx\_val,wy\_val),epochs = 30) | Train model |
| 43 | plt.plot(history.history["accuracy"], c="b")  plt.plot(history.history["val\_accuracy"],c = "r")  plt.grid()  plt.legend(["Training Acc","Validatoin Acc"]) | Plot กราฟดูเรื่อง accuracy ว่า model เรามัน overfitting ไหม |
| 44 | plt.plot(history.history['loss'],c = "b")  plt.plot(history.history["val\_loss"],c ="r")  plt.grid()  plt.legend(["Training Loss","Validatoin Loss"]) | Plot กราฟดูเรื่อง error ว่า model เรามัน underfitting ไหม |
| 45 | model.evaluate(wx\_test,wy\_test) | วัดประสิทธิภาพของmodel |
| 46 | y\_pred = model.predict(wx\_test)  y\_pred = np.argmax(y\_pred,axis=1) | Predict และ argmax คืออะไรไม่รู้ |
| 47 | cm = metric.confusion\_matrix(wy\_test,y\_pred)  plt.figure(figsize = (10,7))  sns.heatmap(cm,cmap = 'YlGnBu',annot = True)  plt.xlabel("Predict")  plt.ylabel("Actual") | สร้างกราฟ confusion matrix |
| 48 | nn\_acc = metric.accuracy\_score(wy\_test,y\_pred)  print("Accuracy: ",nn\_acc) | แสดงค่า acc |
| 49 | #K-Mean Clustering  df = pd.DataFrame(wine.data,columns = wine.feature\_names)  df.head() | ดูข้อมูล 5 อันแรก |
| 50 | x = df[['magnesium','flavanoids']] | ให้ x = ข้อมูล columns magnesium and favanoids |
| 51 | plt.scatter(x['magnesium'],x['flavanoids'],c='black')  plt.xlabel("Magesium")  plt.ylabel("Flabanoids") | Plot จุดเป็นสีดำ |
| 52 | K = 3  centroids = (x.sample(n=K)) | กำหนดจุดในการกลางในการแบ่งกลุ่ม |
| 53 | plt.scatter(x["magnesium"],x['flavanoids'],c='black')  plt.scatter(centroids["magnesium"],centroids["flavanoids"],c="red")  plt.xlabel("Magnesium")  plt.ylabel("Flavanoids") | Plot จุดในการแบ่งกลุ่ม |
| 54 | diff = 1  j = 0 | กำหนดค่าเริ่มต้น |
| 55 | while(diff != 0):  xd = x  i = 1  for index\_1,row\_c in centroids.iterrows():  ed = []  for index\_2,row\_d in xd.iterrows():  d1 = (row\_c['magnesium'] - row\_d['magnesium']) \*\* 2  d2 = (row\_c["flavanoids"] - row\_d['flavanoids']) \*\* 2  d = np.sqrt(d1+d2)  ed.append(d)  x[i] = ed  i = i+1  c = []  for index,row in x.iterrows():  min\_dist = row[1]  pos = 1  for i in range(K):  if row[i+1] < min\_dist:  min\_dist = row[i+1]    pos= i+1  c.append(pos)  x["Cluster"] = c  centroids\_new = x.groupby(["Cluster"]).mean()[["flavanoids","magnesium"]]    if j == 0 :  diff = 1  j = j+1  else:  diff = ((centroids\_new["flavanoids"] - centroids["flavanoids"]).sum() +  (centroids\_new["magnesium"] - centroids["magnesium"]).sum())  print(diff.sum())  centroids = x.groupby(["Cluster"]).mean()[["flavanoids","magnesium"]] | วนรูปการหาจุดกึ่งกลางในการแบ่งกลุ่มที่ดีที่สุด |
| 56 | color = ["blue","green","cyan"]  for k in range(K):  data = x[x["Cluster"] == k+1]  plt.scatter(data["magnesium"],data["flavanoids"],c = color[k])    plt.scatter(centroids["magnesium"],centroids["flavanoids"],c="red")  plt.xlabel("Magnesium")  plt.ylabel("Flavanoids") | Plot กราฟที่แบ่งกลุ่มที่ดีที่สุดแล้ว |
| 57 | #Hierarcy Clustering  from sklearn.cluster import AgglomerativeClustering  from sklearn.datasets import make\_classification  from sklearn.metrics.pairwise import euclidean\_distances  from scipy.cluster import hierarchy | Import library |
| 58 | X,y = make\_classification(1000,n\_features =2,n\_redundant=0) | ไม่แน่ใจ |
| 59 | class HC:  def \_\_init\_\_(self,verbose=False,linkage\_type='conplete'):  self.verbose = verbose  self.linkage\_type = linkage\_type    def argmin(self,D):  minx,miny = (0,0)  min\_val = 10e5  for i in range(D.shape[0]):  for j in range(D.shape[0]):  if j==i:  continue  else:  if D[i,j] < min\_val:  min\_val = D[i,j]  minx = i  miny = j  return min\_val,minx,miny    def cluster\_distance(self,cluster\_members):  nClusters = len(cluster\_members)  keys = list(cluster\_members.keys())  Distance = np.zeros((nClusters,nClusters))  for i in range(nClusters):  ith\_elems = cluster\_members[keys[i]]  for j in range(nClusters):  jth\_elems = cluster\_members[keys[j]]  d\_in\_clusters = euclidean\_distances(X[ith\_elems],X[jth\_elems])  if self.linkage\_type == 'complete':  dij = np.max(d\_in\_clusters)  elif self.linkage\_type == 'single':  dij = np.min(d\_in\_clusters)  Distance[i,j] = dij  return Distance  def fit(self,X):  self.nSamples = X.shape[0]    cluster\_members = dict([(i,[i]) for i in range(self.nSamples)])  Z = np.zeros((self.nSamples-1,4)) #c1 c2 d count  for i in range(0,self.nSamples-1):  nClusters = len(cluster\_members)  keys = list(cluster\_members.keys())  D = self.cluster\_distance(cluster\_members)  \_, tmpx,tmpy = self.argmin(D)    x = keys[tmpx]  y = keys[tmpy]  Z[i,0] = x  Z[i,1] = y  Z[i,2] = D[tmpx,tmpy]  Z[i,3] = len(cluster\_members[x]) + len(cluster\_members[y])    cluster\_members[i+self.nSamples] = cluster\_members[x] + cluster\_members[y]  del cluster\_members[x]  del cluster\_members[y]  self.Z = Z  return self.Z  def predict(self,n\_cluster=3):  labels = np.zeros((self.nSamples))  cluster\_members = dict([(i,[i]) for i in range(self.nSamples)])  for i in range(self.nSamples - n\_cluster):  x,y = (Z[i,0],Z[i,1])  cluster\_members[self.nSamples + i] = cluster\_members[x]+cluster\_members[y]  del cluster\_members[x]  del cluster\_members[y]    keys = list(cluster\_members.keys())    for i in range(len(keys)):  samples\_in\_cluster = cluster\_members[keys[i]]  labels[samples\_in\_cluster] = i  return labels | สร้าง class ของ hierarchy clustering |
| 60 | hc = HC(linkage\_type = "complete")  Z = hc.fit(X)  myLabel = hc.predict(n\_cluster = 3) | สร้าง model  Train model  แบ่งกลุ่มข้อมูลด้วย model |
| 61 | fig,ax = plt.subplots(1,2,figsize=(15,5),dpi =120)  ax = ax.flatten()  for i in range(3):  myIndice = myLabel == i  ax[0].scatter(X[myIndice,0],X[myIndice,1],label = i)  hierarchy.dendrogram(Z,ax = ax[1]) | Plot จุดเป็นสีที่แยก กับ กราฟตารางการแข่งได้ |
| 62 | #Principal Component Analysis  from sklearn.preprocessing import StandardScaler  from sklearn.decomposition import PCA | Import library |
| 63 | labels = np.reshape(wine\_target,(len(wine\_data[:,:3]),1)) | เปลี่ยนมิติของข้อมูล |
| 64 | final\_data = np.concatenate([wine\_data[:,:3], labels],axis=1)  final\_data.shape | จับข้อมูลมาร่วมกัน |
| 65 | df = pd.DataFrame(final\_data)  df.head() | เปลี่ยนข้อมูลให้เป็น dataframe |
|  | features = wine.feature\_names[:3]  features | เรียกชื่อ columns ของ wine มา |
|  | features\_labels = np.append(features,"label") | เพิ่มเข้าไปใน features\_labels |
|  | df.columns = features\_labels  df.head() | แล้วนำไปเปลี่ยนชื่อใน dataframe |
| 66 | fig = px.scatter\_3d(df,x="alcohol",y="malic\_acid",z="ash",color="label")  fig.show() | Plot กราฟเป็น 3D |
| 67 | x = df.loc[:,features].values  x = StandardScaler().fit\_transform(x) | เลือกค่าแบบมีเงื่อนไข  แล้วทำการแปลงข้อมูลอะไรสักอย่าง |
| 68 | print(np.mean(x)) | หาค่าเฉลี่ย |
| 69 | print(np.std(x)) | ค่าการกระจายตัว |
| 70 | feat\_cols = ["feature" + str(i) for i in range(x.shape[1])] | สร้าง columns featre0 … |
| 71 | normalized\_wine = pd.DataFrame(x,columns = feat\_cols)  normalized\_wine.head() | ทำการ normalize ขอ้มูล |
| 72 | pca = PCA(n\_components = 2)  pca\_breast = pca.fit\_transform(x) | สร้าง model  Train model |
| 73 | pca\_df = pd.DataFrame(pca\_breast,columns = ["principal component 1","principal component 2"])  pca\_df.head() | สร้างตารางข้อมูล |
| 74 | print("Explanined varation per principal component:{}".format(pca.explained\_variance\_ratio\_)) | แสดงข้อมูล component ไว้ทำอะไรไม่รู้ |
| 75 | plt.figure(figsize =(10,10))  plt.xticks(fontsize = 12)  plt.yticks(fontsize = 14)  plt.xlabel("Principal Component -1",fontsize = 20)  plt.ylabel("Principal Component -2",fontsize = 20)  plt.title("Principal Component Analysis of Breast Cancer Dataset",fontsize =20)  targets = [0,1,2]  colors = ["r","g","b"]  for target,color in zip(targets,colors):  indiceToKeep = df['label'] == target  plt.scatter(pca\_df.loc[indiceToKeep,"principal component 1"],  pca\_df.loc[indiceToKeep,"principal component 2"],  c = color)  plt.legend(["0","1","2"],prop = {"size":15}) | Plot กราฟ อะไรสักอย่าง |