ML Lab 1

*#%%*

*import numpy as np*

*from sklearn.datasets import make\_regression*

*import matplotlib.pyplot as plt*

***"""***

***"""***

*#%%*

*X , t = make\_regression (100,1, shuffle = True , bias = 0.0 ,noise = 40, random\_state= 4) ;*

***"""***

***"""***

*#%%*

*plt.scatter(X,t)*

***"""***

***"""***

*#%%*

*mean\_x = np.mean(np.squeeze(X))*

*mean\_t = np.mean(t)*

*std\_x = np.std(X)*

*std\_t = np.std(t)*

***"""***

***"""***

*#%%*

*d\_x= X- mean\_x*

*d\_t= t- mean\_t*

*num = np.sum(np.squeeze(d\_x)\*d\_t)*

*deno = np.sum(np.squeeze(d\_x)\*np.squeeze(d\_x))*

*B1= num/ deno*

*B0= mean\_t - B1\*mean\_x*

***"""***

***"""***

*#%%*

*print(B1)*

*print(B0)*

*plt.scatter(X,t)*

***"""***

***"""***

*#%%*

*y= B0 + B1\*X[50,0]*

*print(y)*

*print(t[50])*

*plt.scatter(X,t)*

***"""***

***"""***

ML Lab 2

*import numpy as np*

*from sklearn.datasets import make\_regression*

*#%%*

***"""***

***"""***

*X , t = make\_regression (100,5, shuffle = True , bias = 0.0 , n\_targets=3 ,noise = 10, random\_state= 4) ;*

*#%%*

***"""***

***"""***

*x0=np.ones((100,1), dtype=int);*

*new\_x = np.concatenate((x0,X), axis = 1);*

*tr=new\_x.transpose();*

*m3 = np. dot(tr,new\_x);*

*#%%*

***"""***

***"""***

*i = np.linalg.inv(m3);*

*m4 = np. dot(i,tr);*

*m5=np.dot(m4,t);*

*transp=m5.transpose();*

*#%%*

***"""***

***"""***

*print("the value of W is:-");*

*print(m5);*

*print("value of w using sklearn function")*

*#%%*

***"""***

***"""***

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(new\_x,t)*

*#%%*

***"""***

***"""***

*print(model.intercept\_)*

*print(model.coef\_)*

*#%%*

***"""***

***"""***

ML Lab 3

*from sklearn.datasets import make\_regression*

*X,t = make\_regression(100, 5, shuffle = True, bias = 0, noise = 77, random\_state = 9)*

*X\_mean = X.mean()*

*t\_mean = t.mean()*

*X\_std = X.std()*

*t\_std = t.std()*

*print(X)*

*print(t)*

*import numpy as np*

*x0 = np.ones((100,1))*

*new\_X = np.concatenate((x0, X), axis =1)*

*new\_X.shape*

*X\_t = np.transpose(new\_X)*

*W\_inv = np.dot(X\_t,new\_X)*

*W\_1 = np.linalg.inv(W\_inv)*

*W\_2 = np.dot(X\_t,t)*

*W = np.dot(W\_1, W\_2)*

*print(W)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X,t)*

*model.coef\_*

*#%%*

*X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 70, random\_state = 9, n\_targets = 3)*

*print(X\_vec)*

*print(t\_vec)*

*x0 = np.ones((100,1))*

*new\_X = np.concatenate((x0, X\_vec), axis =1)*

*X\_vec\_t = np.transpose(new\_X)*

*W\_inv = np.dot(X\_vec\_t,new\_X)*

*W\_1 = np.linalg.inv(W\_inv)*

*W\_2 = np.dot(W\_1, X\_vec\_t)*

*W = np.dot(W\_2, t\_vec)*

*print(W)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X\_vec,t\_vec)*

*model.coef\_*

*model.intercept\_*

*#%%*

*from sklearn.metrics import r2\_score*

*y\_predict = model.predict(X\_vec)*

*score = r2\_score(t\_vec, y\_predict)*

*print(score)*

*#%%*

*noise\_lt = [0, 10, 30, 50, 70, 90, 100]*

*r2\_lt = [1.0, 0.9911277146802907, 0.9233227047988551, 0.8082131795660822, 0.6769299465749685, 0.5533416629728133, 0.4983050464146044]*

*import matplotlib.pyplot as plt*

*plt.plot(noise\_lt, r2\_lt)*

*#%%*

*plt.plot(X\_vec,t\_vec)*

*#%%*

*import sklearn.model\_selection as model\_selection*

*X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 100, random\_state = 9, n\_targets = 3)*

*X\_train, X\_test, Y\_train, Y\_test = model\_selection.train\_test\_split(X\_vec, t\_vec, test\_size=0.3)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X\_train,Y\_train)*

*model.coef\_*

*model.intercept\_*

*y\_pred\_train = model.predict(X\_train)*

*score\_train = r2\_score(y\_pred\_train, Y\_train)*

*print(score\_train)*

*#%%*

*y\_pred\_test = model.predict(X\_test)*

*score\_test = r2\_score(y\_pred\_test, Y\_test)*

*print(score\_test)*

*#%%*

*noise\_list = [0, 10, 30, 50, 70 ,90 ,100]*

*score\_train\_list = [1,0.9892422892838316,0.9123535974919882,0.7851886218805498,0.4866233488454797,0.2540158327162871,-0.07161444865168864]*

*score\_test\_list = [1,0.9927037387643658,0.9000725805417417,0.698401043958205,0.5189450019757639,0.2725673734949044,-0.2642270301805036]*

*plt.plot(noise\_list, score\_train\_list)*

*plt.plot(noise\_list, score\_test\_list)*

*fig = plt.figure()*

*ax1 = fig.add\_subplot(221)*

*ax2 = fig.add\_subplot(222)*

*ax1.plot(noise\_list, score\_train\_list)*

*ax2.plot(noise\_list, score\_test\_list)*

ML Lab 4

*from sklearn.datasets import make\_regression*

*import numpy as np*

*X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 70, random\_state = 9, n\_targets = 3)*

*print(X\_vec)*

*print(t\_vec)*

*x0 = np.ones((100,1))*

*new\_X = np.concatenate((x0, X\_vec), axis =1)*

*X\_vec\_t = np.transpose(new\_X)*

*W\_inv = np.dot(X\_vec\_t,new\_X)*

*W\_1 = np.linalg.inv(W\_inv)*

*W\_2 = np.dot(W\_1, X\_vec\_t)*

*W = np.dot(W\_2, t\_vec)*

*print(W)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X\_vec,t\_vec)*

*model.coef\_*

*model.intercept\_*

*#%%*

*from sklearn.metrics import r2\_score*

*y\_predict = model.predict(X\_vec)*

*score = r2\_score(t\_vec, y\_predict)*

*print(score)*

*#%%*

*noise\_lt = [0, 10, 30, 50, 70, 90, 100]*

*r2\_lt = [1.0, 0.9911277146802907, 0.9233227047988551, 0.8082131795660822, 0.6769299465749685, 0.5533416629728133, 0.4983050464146044]*

*import matplotlib.pyplot as plt*

*plt.plot(noise\_lt, r2\_lt)*

*#%%*

*plt.plot(X\_vec,t\_vec)*

*#%%*

*import sklearn.model\_selection as model\_selection*

*X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 100, random\_state = 9, n\_targets = 3)*

*X\_train, X\_test, Y\_train, Y\_test = model\_selection.train\_test\_split(X\_vec, t\_vec, test\_size=0.3)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X\_train,Y\_train)*

*model.coef\_*

*model.intercept\_*

*y\_pred\_train = model.predict(X\_train)*

*score\_train = r2\_score(y\_pred\_train, Y\_train)*

*print(score\_train)*

*#%%*

*y\_pred\_test = model.predict(X\_test)*

*score\_test = r2\_score(y\_pred\_test, Y\_test)*

*print(score\_test)*

*#%%*

*noise\_list = [0, 10, 30, 50, 70 ,90 ,100]*

*score\_train\_list = [1,0.9892422892838316,0.9123535974919882,0.7851886218805498,0.4866233488454797,0.2540158327162871,-0.07161444865168864]*

*score\_test\_list = [1,0.9927037387643658,0.9000725805417417,0.698401043958205,0.5189450019757639,0.2725673734949044,-0.2642270301805036]*

*plt.plot(noise\_list, score\_train\_list)*

*plt.plot(noise\_list, score\_test\_list)*

*fig = plt.figure()*

*ax1 = fig.add\_subplot(221)*

*ax2 = fig.add\_subplot(222)*

*ax1.plot(noise\_list, score\_train\_list)*

*ax2.plot(noise\_list, score\_test\_list)*

ML Lab 5

*#%%*

*import sklearn.model\_selection as model\_selection*

*from sklearn.datasets import make\_regression*

*noise\_ls = [0,5,10,15,20,25,30]*

*score\_train\_list = []*

*score\_test\_list = []*

*score\_tr\_ridge = []*

*score\_ts\_ridge = []*

*score\_tr\_lasso = []*

*score\_ts\_lasso = []*

*score\_tr\_elastic = []*

*score\_ts\_elastic = []*

*for i in noise\_ls:*

*X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = i, random\_state = 9, n\_targets = 3)*

*X\_train, X\_test, Y\_train, Y\_test = model\_selection.train\_test\_split(X\_vec, t\_vec, test\_size=0.3)*

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*w = model.fit(X\_train,Y\_train)*

*model.coef\_*

*model.intercept\_*

*y\_pred\_train = model.predict(X\_train)*

*from sklearn.metrics import r2\_score*

*score\_train = r2\_score(y\_pred\_train, Y\_train)*

*print(score\_train)*

*score\_train\_list.append(score\_train)*

*y\_pred\_test = model.predict(X\_test)*

*score\_test = r2\_score(y\_pred\_test, Y\_test)*

*print(score\_test)*

*score\_test\_list.append(score\_test)*

*import matplotlib.pyplot as plt*

*from sklearn.linear\_model import Ridge*

*model\_ridge = Ridge(alpha=1.0)*

*w\_2 = model\_ridge.fit(X\_train, Y\_train)*

*model\_ridge.coef\_*

*model\_ridge.intercept\_*

*y\_pred\_train\_ridge = model\_ridge.predict(X\_train)*

*score\_train\_ridge = r2\_score(y\_pred\_train\_ridge, Y\_train)*

*print(score\_train\_ridge)*

*score\_tr\_ridge.append(score\_train\_ridge)*

*y\_pred\_test\_ridge = model\_ridge.predict(X\_test)*

*score\_test\_ridge = r2\_score(y\_pred\_test\_ridge, Y\_test)*

*print(score\_test\_ridge)*

*score\_ts\_ridge.append(score\_test\_ridge)*

*from sklearn import linear\_model*

*model\_lasso = linear\_model.Lasso(alpha=0.1)*

*w\_3 = model\_lasso.fit(X\_train, Y\_train)*

*model\_lasso.coef\_*

*model\_lasso.intercept\_*

*y\_pred\_train\_lasso = model\_lasso.predict(X\_train)*

*score\_train\_lasso = r2\_score(y\_pred\_train\_lasso, Y\_train)*

*print(score\_train\_lasso)*

*score\_tr\_lasso.append(score\_train\_lasso)*

*y\_pred\_test\_lasso = model\_lasso.predict(X\_test)*

*score\_test\_lasso = r2\_score(y\_pred\_test\_lasso, Y\_test)*

*print(score\_test\_lasso)*

*score\_ts\_lasso.append(score\_test\_lasso)*

*from sklearn.linear\_model import ElasticNet*

*model\_elastic = ElasticNet(random\_state=0)*

*w\_3 = model\_elastic.fit(X\_train, Y\_train)*

*model\_elastic.coef\_*

*model\_elastic.intercept\_*

*y\_pred\_train\_elastic = model\_elastic.predict(X\_train)*

*score\_train\_elastic = r2\_score(y\_pred\_train\_elastic, Y\_train)*

*print(score\_train\_elastic)*

*score\_tr\_elastic.append(score\_train\_elastic)*

*y\_pred\_test\_elastic = model\_elastic.predict(X\_test)*

*score\_test\_elastic = r2\_score(y\_pred\_test\_elastic, Y\_test)*

*print(score\_test\_elastic)*

*score\_ts\_elastic.append(score\_test\_elastic)*

*plt.plot(noise\_ls, score\_train\_list)*

*plt.plot(noise\_ls, score\_test\_list)*

*plt.plot(noise\_ls, score\_tr\_ridge)*

*plt.plot(noise\_ls, score\_ts\_ridge)*

*plt.plot(noise\_ls, score\_tr\_elastic)*

*plt.plot(noise\_ls, score\_ts\_elastic)*

*plt.plot(noise\_ls, score\_tr\_lasso)*

*plt.plot(noise\_ls, score\_ts\_lasso)*

ML Lab 6

*from sklearn.linear\_model import LogisticRegression*

*import numpy as np*

*from sklearn.datasets import make\_classification*

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

*from sklearn.linear\_model import SGDClassifier*

*#%%*

*classes = 4*

*X, t = make\_classification(100, 5, n\_classes = classes, random\_state= 40, n\_informative = 2, n\_clusters\_per\_class = 1)*

*#%%*

*X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)*

*#%%*

*model = LogisticRegression(random\_state = 1, penalty='none', solver = "saga", max\_iter=4000)*

*#model = LogisticRegression()*

*model.fit(X\_train, t\_train)*

*z\_train = model.predict(X\_train)*

*z\_test = model.predict(X\_test)*

*#%%*

*print(model.coef\_, model.intercept\_)*

*#logistic regression itself generates one hot vector*

*#%%*

*#model\_SGD = SGDClassifier(loss='log')*

*#model\_SGD.fit(X\_train, t\_train)*

*#z\_SGD = model\_SGD.predict(X\_test)*

*#print(model\_SGD.coef\_, model\_SGD.intercept\_)*

*#%%*

*from sklearn.metrics import accuracy\_score*

*final\_score\_train = accuracy\_score(z\_train,t\_train)*

*print(final\_score\_train)*

*final\_score\_test = accuracy\_score(z\_test,t\_test)*

*print(final\_score\_test)*

*#final\_score\_SGD = accuracy\_score(z\_SGD,t\_test)*

*#print(final\_score\_SGD)*

ML Lab 7

Two class

*#%%*

*import numpy as np*

*from sklearn.datasets import make\_classification*

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

*#%%*

*X,t= make\_classification(100, 5, n\_classes = 2, random\_state= 15, n\_informative = 2, n\_clusters\_per\_class = 1)*

*#%%*

*X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)*

*X0=np.ones((80,1))*

*new\_x = np.concatenate((X0,X\_train), axis = 1)*

*X\_transpose = new\_x.transpose()*

*X\_multiply = X\_transpose.dot(new\_x)*

*X\_inverse = np.linalg.inv(X\_multiply)*

*X\_multiply1 = X\_inverse.dot(X\_transpose)*

*w = X\_multiply1.dot(t\_train)*

*#%%*

*Y\_model\_train = new\_x.dot(w)*

*print(Y\_model\_train)*

*#%%*

*x\_0 = np.ones((20,1))*

*new\_x\_test = np.concatenate((x\_0, X\_test), axis = 1)*

*y = np.dot(new\_x\_test, w)*

*print(y)*

*#%%*

*array = []*

*for i in y:*

*if i>=0:*

*array.append(1)*

*else:*

*array.append(0)*

*print(array)*

*#%%*

*from sklearn.metrics import accuracy\_score*

*final\_score = accuracy\_score(array, t\_test)*

*print(final\_score)*

multi class

*#%%*

*import numpy as np*

*from sklearn.datasets import make\_classification*

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

*#%%*

*classes = 3*

*X, t = make\_classification(100, 5, n\_classes = classes, random\_state= 40, n\_informative = 2, n\_clusters\_per\_class = 1)*

*res = np.zeros((t.shape[0], classes), dtype=int)*

*res[np.arange(t.shape[0]), t] = 1*

*print(res)*

*#%%*

*X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)*

*#%%*

*X0 = np.ones((100,1))*

*new\_x = np.concatenate((X0, X), axis = 1)*

*X\_transpose = new\_x.transpose()*

*X\_multiply = X\_transpose.dot(new\_x)*

*X\_inverse = np.linalg.inv(X\_multiply)*

*X\_multiply1 = X\_inverse.dot(X\_transpose)*

*#%%*

*w = X\_multiply1.dot(res)*

*print(w)*

*#%%*

*x\_0 = np.ones((20,1))*

*new\_x\_test = np.concatenate((x\_0, X\_test), axis = 1)*

*y = np.dot(new\_x\_test, w)*

*print(y)*

*#%%*

*z = np.argmax(y, axis = 1)*

*print(z)*

*#%%*

*res\_z = np.zeros((z.shape[0], classes), dtype=int)*

*res\_z[np.arange(z.shape[0]), z] = 1*

*print(res\_z)*

*#%%*

*from sklearn.metrics import accuracy\_score*

*final\_score = accuracy\_score(res\_z, t\_test)*

*print(final\_score)*

Lab 8

*# Decision tree*

*import numpy as np*

*from sklearn.datasets import make\_classification*

*from sklearn.linear\_model import LogisticRegression*

*import matplotlib.pyplot as plt*

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

*from sklearn import tree*

*#%%*

*classes = 3*

*X, t = make\_classification(100, 5, n\_classes = classes, random\_state= 40, n\_informative = 2, n\_clusters\_per\_class = 1)*

*# res = np.zeros((t.shape[0], classes), dtype=int)*

*# res[np.arange(t.shape[0]), t] = 1*

*X\_train, X\_test, t\_train, t\_test = train\_test\_split(X,t)*

*#%%*

*model = tree.DecisionTreeClassifier()*

*model.fit(X\_train, t\_train)*

*tree.plot\_tree(model)*

*"""*

*you may use arguments rom the following*

*DecisionTreeClassifier(\*, criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, class\_weight=None, ccp\_alpha=0.0)[source]¶*

*"""*

Lab 9

*import pandas as pd*

*from sklearn.preprocessing import LabelEncoder*

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

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn import datasets*

*#%% Create a list ‘data’ with the sample dataset*

*data = {'CGPA':['g9','l9','g9','l9','g9'],*

*'Inter':['Y','N','N','N','Y'],*

*'PK':['++','==','==','==','=='],*

*'CS':['G','G','A','A','G'],*

*'Job':['Y','Y','N','N','Y']}*

*#%%*

*#Loading iris dataset*

*iris = datasets.load\_iris()*

*#%%*

*iris\_df = pd.DataFrame(data)*

*#%% Create pandas dataframe “table” using the structure DataFrame with the given dataset*

*table=pd.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job"])*

*#%% Use a value ["CGPA"]=="g9" in the table to select matching row and count the number of columns.*

*table.where(table["CGPA"]=="g9").count()*

*#%% Use LabelEncoder() to encode target labels with value between 0 and no\_of\_classes-1.*

*encoder=LabelEncoder()*

*#%% Then transform non-numerical labels to numerical labels.*

*for i in table:*

*table[i]=encoder.fit\_transform(table[i])*

*print(table)*

*#%% Use iloc property to select by position.*

*X=table.iloc[:,0:4].values*

*t=table.iloc[:,4].values*

*X\_train,X\_test,t\_train,t\_test=train\_test\_split(X,t,test\_size=0.2,random\_state=2)*

*#%% Use RandomForestClassifier class. The most important parameter used is n\_estimators.*

*#model = RandomForestClassifier(n\_estimators=5, max\_depth = 4)*

*#model = RandomForestClassifier(n\_estimators=7, criterion = 'gini')*

*model = RandomForestClassifier(n\_estimators=7, criterion = 'gini', random\_state = 0)*

*X\_iris = iris.data*

*y\_iris = iris.target*

*model.fit(X\_train, t\_train)*

*#%% the fitted model can be used to predict a new instance.*

*# The non-numerical equivalent of the new instance [0, 1, 1, 1] given is [‘g9’, ‘Y’, ‘==’, ‘G’]*

*if model.predict([[0,1,1,1]])==1:*

*print("Got JOB")*

*else:*

*print("Didn't get JOB")*

*#%%*

*if model.predict([[0,0,1,0]])==1:*

*print("Got JOB")*

*else:*

*print("Didn't get JOB")*

*# The non-numerical equivalent of the new instance [0, 0, 1, 0] given is [g9’, ‘N’, ‘==’, ‘A’]*

*#%%*

*print(model.estimators\_)*

*#%%*

*import matplotlib.pyplot as plt*

*from sklearn.tree import plot\_tree*

*from sklearn import tree*

*#%%*

*#plt.figure(figsize=(15, 10))*

*tree.plot\_tree(model.estimators\_[2],*

*filled=True)*

*#%%*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.ensemble import AdaBoostClassifier*

*#%%*

*#model\_cla\_ada = AdaBoostClassifier(n\_estimators=25, random\_state=0)*

*#base = DecisionTreeClassifier(max\_depth = 4)*

*model\_cla\_ada = AdaBoostClassifier(n\_estimators=11, random\_state=0) #, base\_estimator=base)*

*model\_cla\_ada.fit(X\_train, t\_train)*

*#%%*

*print(model\_cla\_ada.estimators\_)*

*#%%*

*tree.plot\_tree(model\_cla\_ada.estimators\_[0],*

*filled=True)*

*#%%*

*if model\_cla\_ada.predict([[0,1,1,1]])==1:*

*print("Got JOB")*

*else:*

*print("Didn't get JOB")*

*#%%*

*if model\_cla\_ada.predict([[1,0,1,1]])==1:*

*print("Got JOB")*

*else:*

*print("Didn't get JOB")*

*#%%*

*from sklearn.datasets import make\_regression*

*from sklearn.ensemble import AdaBoostRegressor*

*from sklearn.metrics import r2\_score*

*#%%*

*X\_reg,t\_reg = make\_regression(150, 3, n\_informative=10, noise=0.3, random\_state=6)*

*X\_train,X\_test,t\_train,t\_test=train\_test\_split(X\_reg,t\_reg,test\_size=0.3,random\_state=2)*

*#%%*

*model\_reg = AdaBoostRegressor()*

*model\_reg.fit(X\_train, t\_train)*

*#%%*

*y\_pred\_train = model\_reg.predict(X\_train)*

*#tree.plot\_tree(model\_reg, feature\_names=X\_train[1], filled=True)*

*plt.plot(X\_train, t\_train, c="r", label="n\_estimators=300", linewidth=2)*

*#%%*

*score\_train = r2\_score(y\_pred\_train, t\_train)*

*print(score\_train)*

*#%%*

*y\_pred\_test = model\_reg.predict(X\_test)*

*score\_test = r2\_score(y\_pred\_test, t\_test)*

*print(score\_test)*

Lab 10

*# -\*- coding: utf-8 -\*-*

*"""*

*Created on Tue Mar 22 12:18:38 2022*

*@author: gopal.singh*

*"""*

*import numpy as np*

*def abs(x):*

*return x if x>0 else -x*

*def sigmoid (x):*

*return 1/(1 + np.exp(-x))*

*def sigmoid\_derivative(x):*

*return x \* (1 - x)*

*def tanh\_act(x):*

*return np.tanh(x)*

*def tanh\_derivative(x):*

*return 1 - x\*\*2*

*def checkError(predicted\_output):*

*expected\_output = [[0],[1],[1],[0]]*

*for i,j in zip(expected\_output , predicted\_output):*

*if abs(i[0]-j[0]) > 0.001:*

*return True*

*return False*

*#Input datasets*

*inputs = np.array([[0,0],[0,1],[1,0],[1,1]])*

*expected\_output = np.array([[0],[1],[1],[0]])*

*epoch = 0*

*lr = 0.1*

*# inputLayerNeurons = 2*

*# hiddenLayerNeurons = 2*

*# outputLayerNeurons = 1*

*inputLayerNeurons = int(input("enter no of inputLayer"))*

*hiddenLayerNeurons = int(input("enter no of hiddenlayer "))*

*outputLayerNeurons = int(input("enter no of outputlayer "))*

*hidden\_weights = []*

*for i in range(1,inputLayerNeurons+1):*

*hidden\_weights\_ind = []*

*for j in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):*

*hidden\_weights\_ind.append(float(input('w'+str(i)+str(j))))*

*hidden\_weights.append(hidden\_weights\_ind)*

*output\_weights = []*

*for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):*

*output\_weights\_ind = []*

*for j in range(inputLayerNeurons+hiddenLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):*

*output\_weights\_ind.append(float(input('w'+str(i)+str(j))))*

*output\_weights.append(output\_weights\_ind)*

*hidden\_bias = []*

*output\_bias = []*

*for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):*

*if i > inputLayerNeurons+hiddenLayerNeurons:*

*output\_bias.append(float(input("o"+str(i))))*

*else:*

*hidden\_bias.append(float(input("o"+str(i))))*

*hidden\_weights = np.asarray(hidden\_weights)*

*hidden\_bias = np.asarray([hidden\_bias])*

*output\_weights = np.asarray(output\_weights)*

*output\_bias = np.asarray([output\_bias])*

*print("Initial hidden weights: ",end='')*

*print(\*hidden\_weights)*

*print("Initial hidden biases: ",end='')*

*print(\*hidden\_bias)*

*print("Initial output weights: ",end='')*

*print(\*output\_weights)*

*print("Initial output biases: ",end='')*

*print(\*output\_bias)*

*predicted\_output = [[0],[0],[0],[0]]*

*#Training algorithm*

*while checkError(predicted\_output):*

*epoch += 1*

*#Forward Propagation*

*hidden\_layer\_activation = np.dot(inputs,hidden\_weights)*

*hidden\_layer\_activation += hidden\_bias*

*hidden\_layer\_output = tanh\_act(hidden\_layer\_activation)*

*output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)*

*output\_layer\_activation += output\_bias*

*predicted\_output = tanh\_act(output\_layer\_activation)*

*#Backpropagation*

*error = expected\_output - predicted\_output*

*d\_predicted\_output = error \* tanh\_derivative(predicted\_output)*

*error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)*

*d\_hidden\_layer = error\_hidden\_layer \* tanh\_derivative(hidden\_layer\_output)*

*#Updating Weights and Biases*

*output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr*

*output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr*

*hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr*

*hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr*

*print("Final hidden weights: ",end='')*

*print(\*hidden\_weights)*

*print("Final hidden bias: ",end='')*

*print(\*hidden\_bias)*

*print("Final output weights: ",end='')*

*print(\*output\_weights)*

*print("Final output bias: ",end='')*

*print(\*output\_bias)*

*print("\nOutput from neural network: ",end='')*

*print(\*predicted\_output)*

*print("\nNo of epochs")*

*print(epoch)*

Lab 11

*# -\*- coding: utf-8 -\*-*

*"""*

*Created on Wed Mar 30 11:08:04 2022*

*@author: shudh*

*"""*

*#%%*

*import numpy as np*

*from sklearn.datasets import make\_classification*

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

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*#%%*

*X,t= make\_classification(500, 5, n\_classes = 3, random\_state= 50, n\_informative = 3, n\_clusters\_per\_class = 1)*

*#%%*

*X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2, random\_state = 40)*

*#%%*

*plt.scatter(X\_train[:, 0], X\_train[:, 1], c = t\_train, marker='.')*

*plt.show()*

*#%%*

*from sklearn import svm*

*model\_svm = svm.SVC(kernel = "linear")*

*model\_svm.fit(X\_train, t\_train)*

*y\_pred = model\_svm.predict(X\_test)*

*#%%*

*from sklearn.metrics import accuracy\_score*

*final\_score = accuracy\_score(t\_test, y\_pred)*

*print(final\_score)*

*#%%*

*import random*

*import numpy as np*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*def generate\_random\_dataset(size):*

*""" Generate a random dataset and that follows a quadratic distribution*

*"""*

*x = []*

*y = []*

*target = []*

*for i in range(size):*

*# class zero*

*x.append(np.round(random.uniform(0, 2.5), 1))*

*y.append(np.round(random.uniform(0, 20), 1))*

*target.append(0) # class one*

*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(data=x)*

*df\_y = pd.DataFrame(data=y)*

*df\_target = pd.DataFrame(data=target)*

*data\_frame = pd.concat([df\_x, df\_y], ignore\_index=True, axis=1)*

*data\_frame = pd.concat([data\_frame, df\_target], ignore\_index=True, axis=1)*

*data\_frame.columns = ['x', 'y', 'target']*

*return data\_frame*

*# Generate dataset*

*size = 100*

*dataset = generate\_random\_dataset(size)*

*features = dataset[['x', 'y']]*

*label = dataset['target']# Hold out 20% of the dataset for training*

*test\_size = int(np.round(size \* 0.2, 0))# Split dataset into training and testing sets*

*x\_train = features[:-test\_size].values*

*y\_train = label[:-test\_size].values*

*x\_test = features[-test\_size:].values*

*y\_test = label[-test\_size:].values*

*#%%*

*fig, ax = plt.subplots(figsize=(12, 7))# removing to and right border*

*ax.spines['top'].set\_visible(False)*

*ax.spines['left'].set\_visible(False)*

*ax.spines['right'].set\_visible(False)# adding major gridlines*

*ax.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)*

*ax.scatter(features[:-test\_size]['x'], features[:-test\_size]['y'], color="#8C7298")*

*plt.show()*

*#%%*

*from sklearn import svm*

*model = svm.SVC(kernel='linear')*

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

*#%%*

*fig, ax = plt.subplots(figsize=(12, 7))# Removing to and right border*

*ax.spines['top'].set\_visible(False)*

*ax.spines['left'].set\_visible(False)*

*ax.spines['right'].set\_visible(False)# Create grid to evaluate model*

*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'])# Assigning different colors to the classes*

*colors = y\_train*

*colors = np.where(colors == 1, '#8C7298', '#4786D1')# Plot the dataset*

*ax.scatter(features[:-test\_size]['x'], features[:-test\_size]['y'], c=colors)# Get the separating hyperplane*

*Z = model.decision\_function(xy).reshape(XX.shape)*

*# Draw the decision boundary and margins*

*ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])# Highlight support vectors with a circle around them*

*ax.scatter(model.support\_vectors\_[:, 0], model.support\_vectors\_[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')*

*plt.show()*

Lab 12

*#%%*

*# importing the required libraries*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*#%%*

*# using the make\_blobs dataset*

*from sklearn.datasets import make\_blobs*

*X, y = make\_blobs(n\_samples=100, centers=5, random\_state=101)*

*#%%*

*# setting the number of training examples*

*m=X.shape[0]*

*n=X.shape[1]*

*n\_iter=50*

*#%%*

*# computing the initial centroids randomly*

*K=5*

*import random*

*# creating an empty centroid array*

*centroids=np.array([]).reshape(n,0)*

*# creating 5 random centroids*

*for k in range(K):*

*centroids=np.c\_[centroids,X[random.randint(0,m-1)]]*

*#%%*

*output={}*

*# creating an empty array*

*euclid=np.array([]).reshape(m,0)*

*# finding distance between for each centroid*

*for k in range(K):*

*dist=np.sum((X-centroids[:,k])\*\*2,axis=1)*

*euclid=np.c\_[euclid,dist]*

*# storing the minimum value we have computed*

*minimum=np.argmin(euclid,axis=1)+1*

*#%%*

*# computing the mean of separated clusters*

*cent={}*

*for k in range(K):*

*cent[k+1]=np.array([]).reshape(2,0)*

*# assigning of clusters to points*

*for k in range(m):*

*cent[minimum[k]]=np.c\_[cent[minimum[k]],X[k]]*

*for k in range(K):*

*cent[k+1]=cent[k+1].T*

*# computing mean and updating it*

*for k in range(K):*

*centroids[:,k]=np.mean(cent[k+1],axis=0)*

*#%%*

*# repeating the above steps again and again*

*for i in range(n\_iter):*

*euclid=np.array([]).reshape(m,0)*

*for k in range(K):*

*dist=np.sum((X-centroids[:,k])\*\*2,axis=1)*

*euclid=np.c\_[euclid,dist]*

*C=np.argmin(euclid,axis=1)+1*

*cent={}*

*for k in range(K):*

*cent[k+1]=np.array([]).reshape(2,0)*

*for k in range(m):*

*cent[C[k]]=np.c\_[cent[C[k]],X[k]]*

*for k in range(K):*

*cent[k+1]=cent[k+1].T*

*for k in range(K):*

*centroids[:,k]=np.mean(cent[k+1],axis=0)*

*final=cent*

*#%%*

*plt.scatter(X[:,0],X[:,1])*

*plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})*

*plt.title('Original Dataset')*

*#%%*

*for k in range(K):*

*plt.scatter(final[k+1][:,0],final[k+1][:,1])*

*plt.scatter(centroids[0,:],centroids[1,:],s=300,c='yellow')*

*plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})*

*plt.show()*

*#%%*

*#Elbow MEthod for finding optimal number of clusters*

*from sklearn.datasets import make\_blobs*

*X, y = make\_blobs(n\_samples=100, centers=5, random\_state=101)*

*#%%*

*import seaborn as sns*

*from sklearn.cluster import KMeans*

*elbow=[]*

*for i in range(1, 20):*

*kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 101)*

*kmeans.fit(X)*

*elbow.append(kmeans.inertia\_)*

*sns.lineplot(range(1, 20), elbow,color='blue')*

*plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})*

*plt.title('ELBOW METHOD')*

*plt.show()*

*#%%*