**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR.NO** | **DATE** | **LAB** | **PAGE NO** |
| 1 | 21/02/23 | Gradient Descent for Linear Regression | 3 |
| 2 | 28/02/23 | Gradient Descent for Logistic Regression | 7 |
| 3 | 07/03/23 | Boosting Algorithm | 9 |
| 4 | 14/03/23 | Random Forest | 12 |
| 5 | 21/03/23 | Stacking With Hard Voting | 14 |
| 6 | 28/03/23 | Stacking With Soft Voting | 17 |
| 7 | 04/04/23 | PCA | 20 |
| 8 | 04/04/23 | LDA | 22 |
| 9 | 04/04/23 | SVD | 25 |
| 10 | 11/04/23 | Artificial Neural Network | 27 |

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **21/02/2023** | **1** | To calculate the gradients using cost function and perform gradient decent for linear regression |

**Code:**

import random

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

study\_time=[]

score=[]

for i in range(1,100):

n1=random.randint(1,20)

n2=random.randint(30,100)

study\_time.append(n1)

score.append(n2)

student\_data=pd.DataFrame({'study\_time':study\_time,'score':score})

df=student\_data

plt.scatter(df['study\_time'],df['score'])

plt.show()

std\_norm=df

df\_norm=df

std\_norm['study\_time']=(std\_norm['study\_time'] -(std\_norm['study\_time']).mean())/std\_norm['study\_time'].std()

df\_norm['study\_time']=(df['study\_time']-min(df['study\_time']))/(max(df['study\_time'])-min(df['study\_time']))

#df\_norm['study\_time']=(df['study\_time']-df['study\_time'].mean())/(max(df['study\_time'])-min(df['study\_time']))

plt.scatter(df\_norm['study\_time'],df\_norm['score'])

plt.show()

def loss\_function(m,b,df):

total\_error=0

for i in range(len(df)):

x=df.iloc[i][0]

y=df.iloc[i][1]

total\_error+=(y-(m\*x)+b)\*\*2

return (total\_error/2\*len(df))

def gradient\_descent(m\_now,b\_now,df,L):

n=len(df)

s1=0

s2=0

for i in range(n):

x=df.iloc[i][0]

y=df.iloc[i][1]

s1+=(((m\*x)+b)-y)\*x

s2+=(((m\*x)+b)-y)

m\_now-=L\*s1\*(1/n)

b\_now-=L\*s2\*(1/n)

return m\_now,b\_now

loss=loss\_function(0.5,1.5,df\_norm)

m=0 ,b=0 ,L=0.0001

epochs=100

for i in range(epochs):

m,b=gradient\_descent(m,b,df\_norm,L)

loss=loss\_function(m,b,df\_norm)

print(m,b,loss)

plt.scatter(df\_norm['study\_time'],df\_norm['score'])

test\_x=[]

predicted=[(m\*x)+b for x in df\_norm['study\_time']]

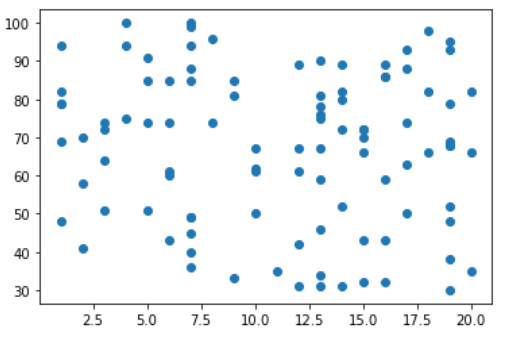
for i in range(80,100):

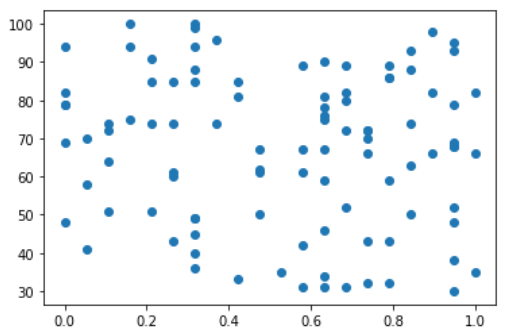
test\_x.append(random.random())

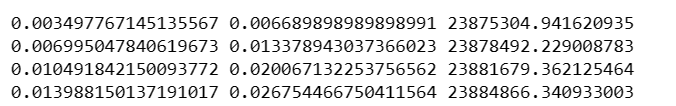
plt.plot(list(range(80,100)),[(m\*x)+b for x in list(range(80,100))])

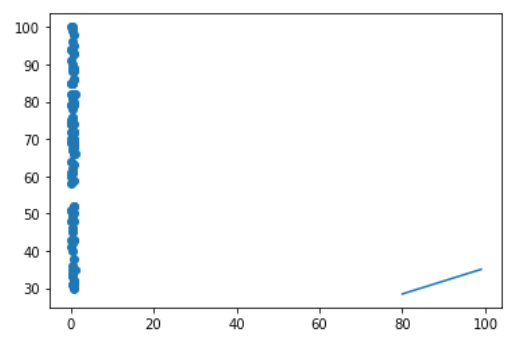
print(predicted)

**Output:**

****

****

****

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **28/02/2023** | **2** | To calculate the gradients using cost function and perform gradient decent for logisitc regression |

**Code:**

import numpy as np

from numpy import log,dot,exp,shape

import matplotlib.pyplot as plt

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

from sklearn.datasets import make\_classification

x,y = make\_classification(n\_features=4)

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.1,random\_state=0)

class LogisticRegression:

def sigmoid(self,z):

sig = 1/(1+exp(-z)) return sig

def initialize(self,X):

weights = np.zeros((shape(X)[1]+1,1))

X = np.c\_[np.ones((shape(X)[0],1)),X] return weights,X

def fit(self,X,y,alpha=0.001,iter=400):

weights,X = self.initialize(X)

def cost(theta):

z = dot(X,theta)

cost0 = y.T.dot(log(self.sigmoid(z)))

cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))

cost = -((cost1 + cost0))/len(y)return cost

cost\_list = np.zeros(iter,)

for i in range(iter):

weights = weights - alpha\*dot(X.T,self.sigmoid(dot(X,weights))-np.reshape(y,(len(y),1)))

cost\_list[i] = cost(weights)

self.weights = weights return cost\_list

def predict(self,X):

z = dot(self.initialize(X)[1],self.weights)

lis = []

for i in self.sigmoid(z):

if i>0.5:

lis.append(1)

else:

lis.append(0) return lis

lgr = LogisticRegression()

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

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

y\_train = lgr.predict(x\_train)

print(y\_pred)

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **07/03/2023** | **3** | To implement AdaBoost and XGBoost algorithms |

**Code:**

**1.AdaBoost**

import pandas as pd

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import mean\_squared\_error, accuracy\_score,confusion\_matrix

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

from sklearn.preprocessing import LabelEncoder

df=pd.read\_csv("Fish.csv")

x=df.drop('Species',axis=1) y=df['Species']

le=LabelEncoder() y=le.fit\_transform(y)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

ada = AdaBoostClassifier(base\_estimator=DecisionTreeClassifier(max\_depth=6),n\_estimators=50,learning\_rate=0.1)

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

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

cm=confusion\_matrix(y\_pred,y\_test)

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

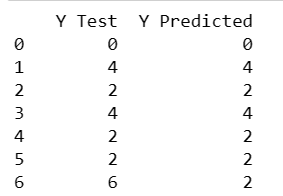
pred=pd.DataFrame(data={'Y Test':list(y\_test), 'Y Predicted':list(y\_pred)})

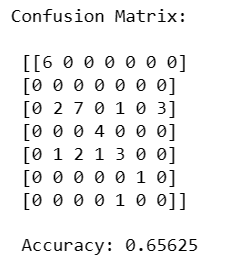
print(pred)

print("Confusion Matrix: \n\n",cm) print("\n Accuracy:", accuracy)

**Output:**

**1.AdaBoost:**

****

****

**2.XGBoost:**

import pandas as pd

import xgboost as xgb

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

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

from sklearn.metrics import confusion\_matrix,accuracy\_score

df=pd.read\_csv("advertising.csv")

x=df.drop('Clicked on Ad',axis=1) y=df['Clicked on Ad']

le=LabelEncoder()

cols=["Ad Topic Line","City","Male","Country","Timestamp"]

for c in cols:

x[c]=le.fit\_transform(x[c])

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

xgb\_model=xgb.XGBClassifier(learning\_rate=0.1,max\_depth=6,n\_estimators=120)

xgb\_model.fit(x\_train,y\_train)

y\_pred=xgb\_model.predict(x\_test)

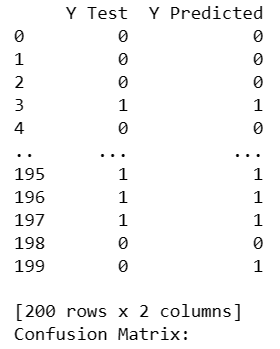
cm=confusion\_matrix(y\_pred,y\_test)

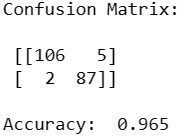
pred=pd.DataFrame(data={'Y Test':list(y\_test), 'Y Predicted':list(y\_pred)})

print(pred) print("Confusion Matrix: \n\n",cm)

print("\nAccuracy: ",accuracy\_score(y\_test,y\_pred))

**Output:**

****

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **14/03/2023** | **4** | To implement Random Forest in ensemble machine learning |

**Code:**

import random

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import accuracy\_score,r2\_score,confusion\_matrix

from sklearn.preprocessing import LabelEncoder

from sklearn import datasets

titanic\_data = pd.read\_csv('https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv')

X = titanic\_data.drop('Survived', axis=1)

y = titanic\_data['Survived']

columns=['Name','Sex']

for col in X.columns:

if(col in columns):

le=LabelEncoder()

X[col]=le.fit\_transform(X[col])

train\_X, test\_X, train\_y, test\_y = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rf=RandomForestClassifier(n\_estimators=100,max\_depth=7)

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

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

cm=confusion\_matrix(y\_pred,y\_test)

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

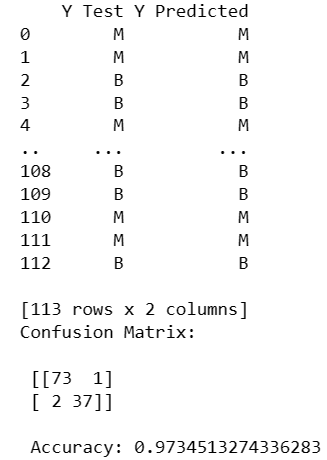
pred=pd.DataFrame(data={'Y Test':list(y\_test), 'Y Predicted':list(y\_pred)})

print(pred)

print("Confusion Matrix: \n\n",cm)

print("\n Accuracy:", accuracy)

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **21/03/2023** | **5** | To implement stacking model with hard voting method and interpret the performance measure |

**Code:**

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import accuracy\_score

import pandas as pd

# Load the Iris dataset

from sklearn.datasets import load\_digits

X, y = load\_digits(return\_X\_y=True)

# Split data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the base classifiers

clf1 = DecisionTreeClassifier(max\_depth=2, random\_state=42)

clf2 = LogisticRegression(random\_state=42)

clf3 = GaussianNB()

clf4 = RandomForestClassifier(random\_state=42)

# Train the base classifiers on the training data

clf1.fit(X\_train, y\_train)

clf2.fit(X\_train, y\_train)

clf3.fit(X\_train, y\_train)

clf4.fit(X\_train, y\_train)

# Make predictions on the test data using the base classifiers

y\_pred1 = clf1.predict(X\_test)

y\_pred2 = clf2.predict(X\_test)

y\_pred3 = clf3.predict(X\_test)

y\_pred4 = clf4.predict(X\_test)

# Combine the predictions of the base classifiers into a stacked dataset

stacked\_data = []

for i in range(len(X\_test)):

stacked\_data.append([y\_pred1[i], y\_pred2[i]])

# Train a meta-classifier on the stacked dataset

meta\_clf = DecisionTreeClassifier(max\_depth=2, random\_state=42)

meta\_clf.fit(stacked\_data, y\_test)

# Make predictions on the test data using the meta-classifier

y\_pred\_meta = meta\_clf.predict(stacked\_data)

# Combine the predictions of the base classifiers and the meta-classifier using hard voting

y\_pred = []

for i in range(len(X\_test)):

votes = [y\_pred1[i], y\_pred2[i], y\_pred\_meta[i]]

y\_pred.append(max(set(votes), key=votes.count))

# Print the accuracy score of the ensemble classifier

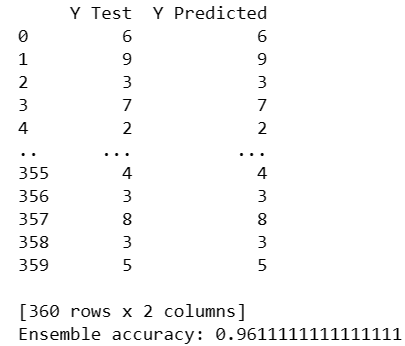
accuracy = accuracy\_score(y\_test, y\_pred)

pred=pd.DataFrame(data={'Y Test':list(y\_test), 'Y Predicted':list(y\_pred)})

print(pred)

print("Ensemble accuracy:", accuracy)

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **28/02/2023** | **6** | To implement stacking model with soft voting method and interpret the performance measure |

**Code:**

from sklearn.datasets import load\_digits

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

# Load the digits dataset

digits = load\_digits()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, digits.target, test\_size=0.2, random\_state=42)

# Define the base classifiers

clf1 = DecisionTreeClassifier(max\_depth=2, random\_state=42)

clf2 = KNeighborsClassifier(n\_neighbors=5)

clf3 = GaussianNB()

clf4 = RandomForestClassifier(n\_estimators=10, random\_state=42)

# Train the base classifiers on the training data

clf1.fit(X\_train, y\_train)

clf2.fit(X\_train, y\_train)

clf3.fit(X\_train, y\_train)

clf4.fit(X\_train, y\_train)

# Make predictions on the test data using the base classifiers

y\_pred1 = clf1.predict\_proba(X\_test)

y\_pred2 = clf2.predict\_proba(X\_test)

y\_pred3 = clf3.predict\_proba(X\_test)

y\_pred4 = clf4.predict\_proba(X\_test)

# Combine the predictions of the base classifiers into a stacked dataset

stacked\_data = np.concatenate((y\_pred1, y\_pred2, y\_pred3, y\_pred4), axis=1)

# Train a meta-classifier on the stacked dataset

meta\_clf = LogisticRegression(random\_state=42)

meta\_clf.fit(stacked\_data, y\_test)

# Make predictions on the test data using the base classifiers and the meta-classifier

y\_pred1 = clf1.predict\_proba(X\_test)

y\_pred2 = clf2.predict\_proba(X\_test)

y\_pred3 = clf3.predict\_proba(X\_test)

y\_pred4 = clf4.predict\_proba(X\_test)

stacked\_data = np.concatenate((y\_pred1, y\_pred2, y\_pred3, y\_pred4), axis=1)

y\_pred\_meta = meta\_clf.predict\_proba(stacked\_data)

# Combine the predictions of the base classifiers and the meta-classifier using soft voting

y\_pred = np.argmax(y\_pred\_meta, axis=1)

# Print the accuracy score of the ensemble classifier

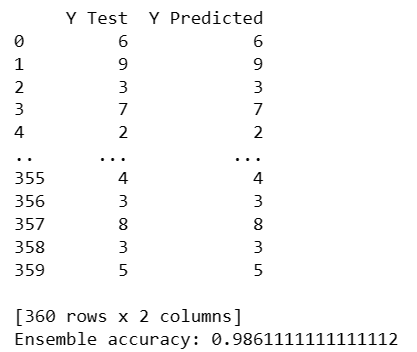
accuracy = accuracy\_score(y\_test, y\_pred)

pred=pd.DataFrame(data={'Y Test':list(y\_test), 'Y Predicted':list(y\_pred)})

print(pred)

print("Ensemble accuracy:", accuracy)

**Output:**



|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **04/04/2023** | **7** | To perform dimensionality reduction using Principal Component Analysis and infer the performance |

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder

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

# load iris dataset

df = datasets.load\_digits()

x = df.data

y = df.target

le=LabelEncoder()

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

sc=StandardScaler()

x\_train=sc.fit\_transform(x\_train,y\_train)

x\_test=sc.transform(x\_test)

# perform PCA

pca = PCA(n\_components=4)

x\_train = pca.fit\_transform(x\_train)

x\_test=pca.transform(x\_test)

explained\_variance=pca.explained\_variance\_ratio\_

print("Explained Variance Ratio: ",explained\_variance)

classifier=RandomForestClassifier(max\_depth=2,random\_state=0)

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

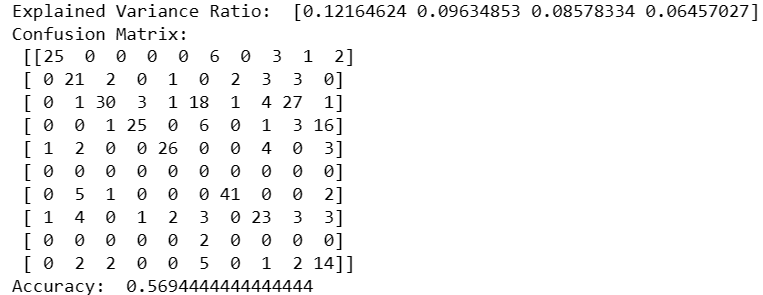
y\_pred=classifier.predict(x\_test)

cm=confusion\_matrix(y\_pred,y\_test)

print("Confusion Matrix: \n",cm)

print("Accuracy: ",accuracy\_score(y\_test,y\_pred))

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **04/04/2023** | **8** | To perform dimensionality reduction using Linear Discriminant Analysis and infer the performance |

**Code:**

import numpy as np

import pandas as pd

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix,accuracy\_score

# load iris dataset

df = datasets.load\_breast\_cancer()

x=df.data y=df.target

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x\_train=sc.fit\_transform(x\_train,y\_train)

x\_test=sc.transform(x\_test)

class LDA:

def \_\_init\_\_(self, n\_components=None):

self.n\_components = n\_components

self.eig\_vectors = None

def transform(self,X,y):

height, width = X.shape

unique\_classes = np.unique(y)

num\_classes = len(unique\_classes)

scatter\_t = np.cov(X.T)\*(height - 1)

scatter\_w = 0

for i in range(num\_classes):

class\_items = np.flatnonzero(y == unique\_classes[i])

scatter\_w = scatter\_w + np.cov(X[class\_items].T) \* (len(class\_items)-1)

scatter\_b = scatter\_t - scatter\_w

\_, eig\_vectors = np.linalg.eigh(np.linalg.pinv(scatter\_w).dot(scatter\_b))

print(eig\_vectors.shape)

pc = X.dot(eig\_vectors[:,::-1][:,:self.n\_components]) print(pc.shape)

if self.n\_components == 2:

if y is None:

plt.scatter(pc[:,0],pc[:,1])

else:

colors = ['r','g','b'], labels = np.unique(y)

for color, label in zip(colors, labels):

class\_data = pc[np.flatnonzero(y==label)]

plt.scatter(class\_data[:,0],class\_data[:,1],c=color)

plt.show() return pc

LDA\_obj = LDA(n\_components=2)

LDA\_object = LDA(n\_components=2)

x\_train\_modified = LDA\_object.transform(x\_train, y\_train)

x\_test\_modified = LDA\_object.transform(x\_test, y\_test)

print("Original Data Size:",x\_train.shape, "\nModified Data Size:", x\_train\_modified.shape) print(x\_train\_modified)

classifier=RandomForestClassifier(max\_depth=2,random\_state=0)

classifier.fit(x\_train\_modified,y\_train)

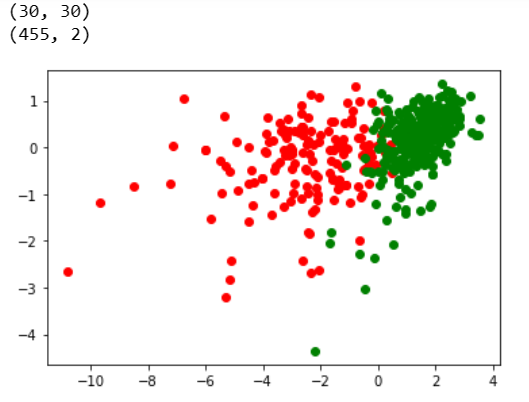
y\_pred=classifier.predict(x\_test\_modified)

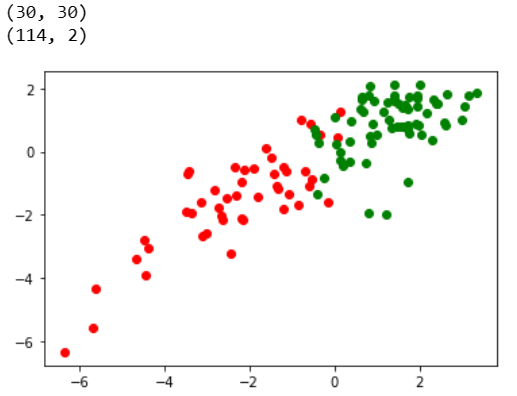
cm=confusion\_matrix(y\_pred,y\_test)

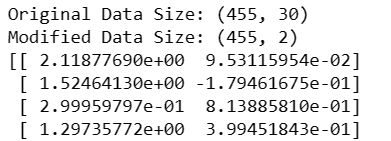
print("Confusion Matrix: \n",cm) print("Accuracy: ",accuracy\_score(y\_test,y\_pred))

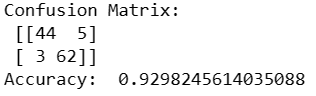
**Output:**

**Variation of Data:**

****

****

****

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **04/04/2023** | **9** | To perform dimensionality reduction using Linear Discriminant Analysis and infer the performance |

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.decomposition import PCA

# load iris dataset

df = datasets.load\_wine()

x = df.data

y = df.target

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x\_train=sc.fit\_transform(x\_train,y\_train)

x\_test=sc.transform(x\_test)

from sklearn.decomposition import TruncatedSVD

svd=TruncatedSVD(n\_components=10,n\_iter=10)

x\_train=svd.fit\_transform(x\_train)

x\_test=svd.transform(x\_test)

explained\_variance=svd.explained\_variance\_ratio\_

print(explained\_variance)

from sklearn.ensemble import RandomForestClassifier

classifier=RandomForestClassifier(max\_depth=2,random\_state=0)

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

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

from sklearn.metrics import confusion\_matrix

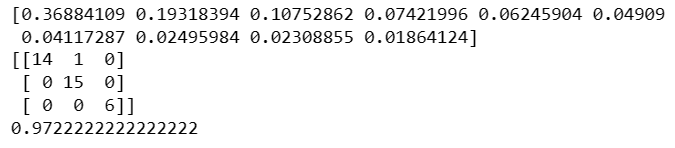
from sklearn.metrics import accuracy\_score

cm=confusion\_matrix(y\_pred,y\_test)

print(cm)

print(accuracy\_score(y\_test,y\_pred))

**Output:**

****

|  |  |  |
| --- | --- | --- |
| **Date** | **Lab No** | **Problem** |
| **11/04/2023** | **10** | To build an Artificial Neural Network that minimizes loss and maximizes the performance |

**Code:**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow import keras

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

from keras.models import Sequential

from keras.layers import Dense

from tensorflow.keras.optimizers import Adam

# Load the data

df = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data',header=None)

x=df.iloc[:,2:]

y=df.iloc[:,1]

y=np.where(y=='M',1,0)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=0)

ann= Sequential()

ann.add(Dense(16, input\_dim=X\_train.shape[1], activation='relu'))

ann.add(Dense(8, activation='relu'))

ann.add(Dense(1, activation='sigmoid'))

# Compile the model

ann.compile(optimizer='adam',loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

ann.fit(X\_train, y\_train, epochs=50, batch\_size=100,verbose=0,validation\_split=0.2)

# Evaluate the model

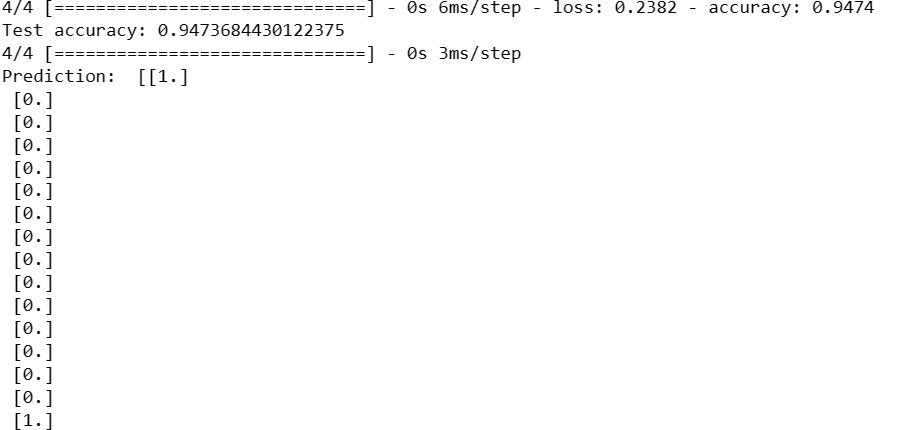
loss, accuracy =ann.evaluate(X\_test, y\_test)

print('Test accuracy:', accuracy)

ann\_predict = ann.predict(X\_test)

print("Prediction: ",np.round(ann\_predict))

**Output:**

****