INDEX

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

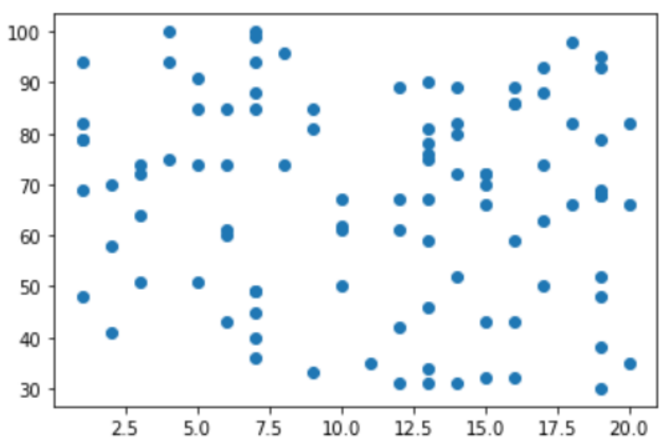
Code:

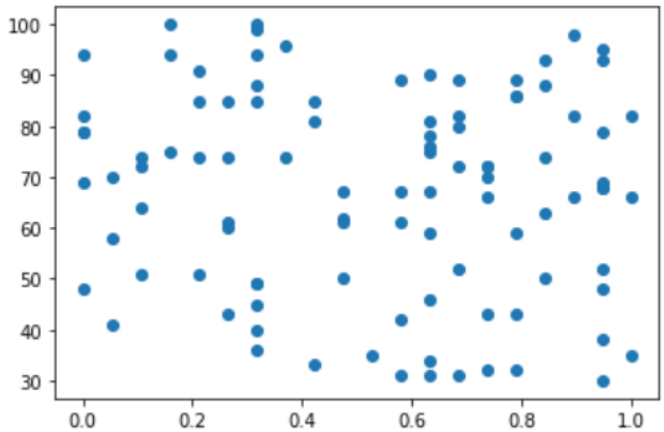
```
import random

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
```

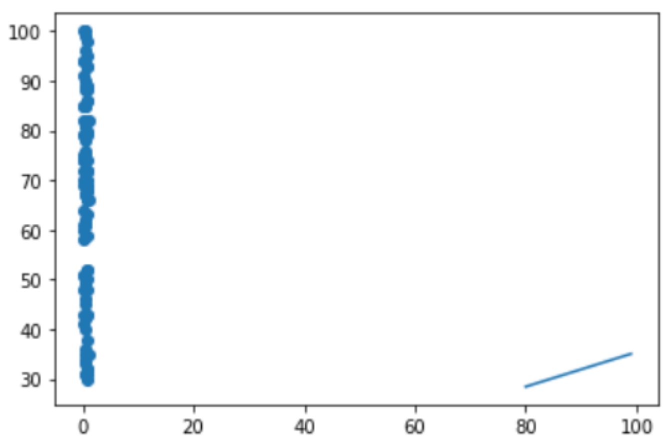
```
loss=loss_function(0.5,1.5,df_norm)
m=0 ,b=0 ,L=0.0001
epochs=100
for i in range(epochs):
    m,b=gradient_descent(m,b,df_norm,L)
    loss=loss_function(m,b,df_norm)
    print(m,b,loss)
plt.scatter(df_norm['study_time'],df_norm['score'])
test_x=[]
predicted=[(m*x)+b for x in df_norm['study_time']]
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:






```
0.003497767145135567 0.006689898989898991 23875304.941620935
0.006995047840619673 0.013378943037366023 23878492.229008783
0.010491842150093772 0.020067132253756562 23881679.362125464
0.013988150137191017 0.026754466750411564 23884866.340933003
```



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:

```
[1, 1, 0, 0, 1, 1, 0, 1, 1, 1]
```

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,l
earning_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:

```
   Y Test  Y Predicted
0       0            0
1       4            4
2       2            2
3       4            4
4       2            2
5       2            2
6       6            2

Confusion Matrix:

 [[6 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 2 7 0 1 0 3]
 [0 0 0 4 0 0 0]
 [0 1 2 1 3 0 0]
 [0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0]]

 Accuracy: 0.65625
```

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:

```
      Y Test  Y Predicted
0          0            0
1          0            0
2          0            0
3          1            1
4          0            0
..       ...          ...
195        1            1
196        1            1
197        1            1
198        0            0
199        0            1

[200 rows x 2 columns]
Confusion Matrix:
```

```
Confusion Matrix:

 [[106   5]
 [  2  87]]

Accuracy:  0.965
```

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:

```
    Y Test Y Predicted
0        M           M
1        M           M
2        B           B
3        B           B
4        M           M
..     ...         ...
108      B           B
109      B           B
110      M           M
111      M           M
112      B           B

[113 rows x 2 columns]
Confusion Matrix: 

 [[73  1]
 [ 2 37]]

 Accuracy: 0.9734513274336283
```

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:

```
     Y Test  Y Predicted
0         6            6
1         9            9
2         3            3
3         7            7
4         2            2
..      ...          ...
355       4            4
356       3            3
357       8            8
358       3            3
359       5            5

[360 rows x 2 columns]
Ensemble accuracy: 0.9611111111111111
```

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:

```
     Y Test  Y Predicted
0         6            6
1         9            9
2         3            3
3         7            7
4         2            2
..      ...          ...
355       4            4
356       3            3
357       8            8
358       3            3
359       5            5

[360 rows x 2 columns]
Ensemble accuracy: 0.9861111111111112
```

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:

```
Explained Variance Ratio:  [0.12164624 0.09634853 0.08578334 0.06457027]
Confusion Matrix:
 [[25  0  0  0  0  6  0  3  1  2]
 [ 0 21  2  0  1  0  2  3  3  0]
 [ 0  1 30  3  1 18  1  4 27  1]
 [ 0  0  1 25  0  6  0  1  3 16]
 [ 1  2  0  0 26  0  0  4  0  3]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  5  1  0  0  0 41  0  0  2]
 [ 1  4  0  1  2  3  0 23  3  3]
 [ 0  0  0  0  0  2  0  0  0  0]
 [ 0  2  2  0  0  5  0  1  2 14]]
Accuracy:  0.5694444444444444
```

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

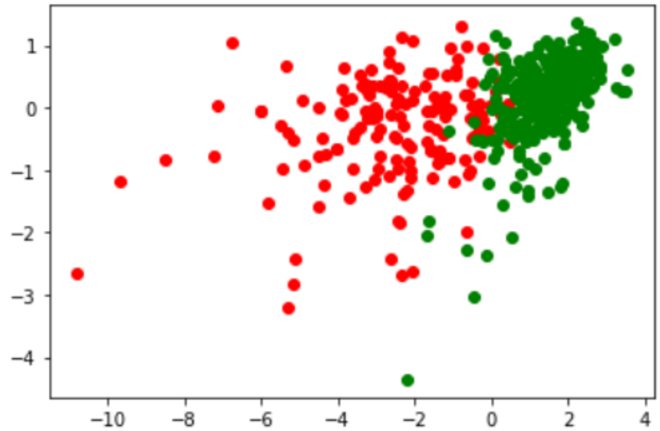
Code:

```
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
```

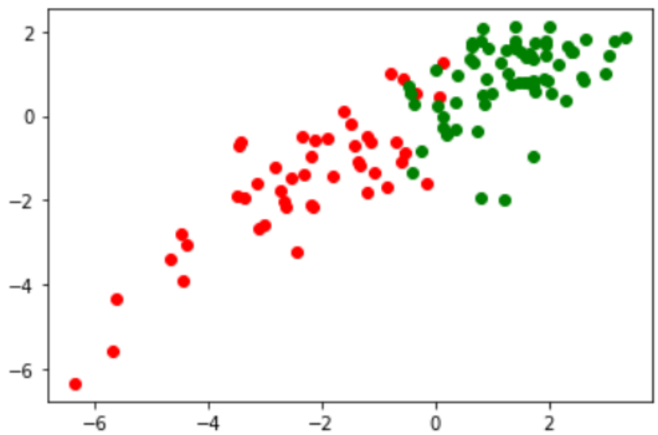
```
        for i in range(num_classes):
            class_items = np.flatnonzero(y == unique_classes[i])
            scatter_w = scatter_w + np.cov(X[class_items].T) * (len(class_items)-1)
        scatter_b = scatter_t - scatter_w
        _, eig_vectors = np.linalg.eigh(np.linalg.pinv(scatter_w).dot(scatter_b))
        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:

```
(30, 30)
(455, 2)
```




```
(30, 30)
(114, 2)
```




```
Original Data Size: (455, 30)
Modified Data Size: (455, 2)
[[ 2.11877690e+00  9.53115954e-02]
 [ 1.52464130e+00 -1.79461675e-01]
 [ 2.99959797e-01  8.13885810e-01]
 [ 1.29735772e+00  3.99451843e-01]]


Confusion Matrix:
 [[44  5]
 [ 3 62]]
Accuracy:  0.9298245614035088
```

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:

```
[0.36884109 0.19318394 0.10752862 0.07421996 0.06245904 0.04909
 0.04117287 0.02495984 0.02308855 0.01864124]
[[14  1  0]
 [ 0 15  0]
 [ 0  0  6]]
0.9722222222222222
```

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:

```
4/4 [==============================] - 0s 6ms/step - loss: 0.2382 - accuracy: 0.9474
Test accuracy: 0.9473684430122375
4/4 [==============================] - 0s 3ms/step
Prediction:  [[1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
```