RAJALAKSHMIENGINEERINGCOLLEGE

[AUTONOMOUS]

RAJALAKSHMINAGAR,THANDALAM-602105

Laboratory Record Note Book

Name : Manasseh Jayanand D

Year/Branch/Section : 1V/CSD/A

Register No : 211701029

College Roll No : 2116211701029

Semester: 7

Academic Year: 2023 - 2024

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMINAGAR,THANDALAM-602105

BONAFIDE CERTIFICATE

Name : MANASSEH JAYANAND D

Academic Year :2023- 2024 Semester : 7 Branch : CSD

RegisterNo: 211701029

Certified that is the bonafide record of work done by the above student in the Foundations of Machine Learning Laboratory during the year 2023 - 2024

Signature of Faculty in-charge

Submitted for the practical examination held on 25.11.2024

Internal Examiner ExternalExaminer

RAJALAKSHMI ENGINEERING COLLEGE

INDEX

EXPT NO: 01 LINEAR REGRESSION

DATE:26/7/2024

AIM:

To predict continuous target values using the Linear Regression algorithm.

ALGORITHM:

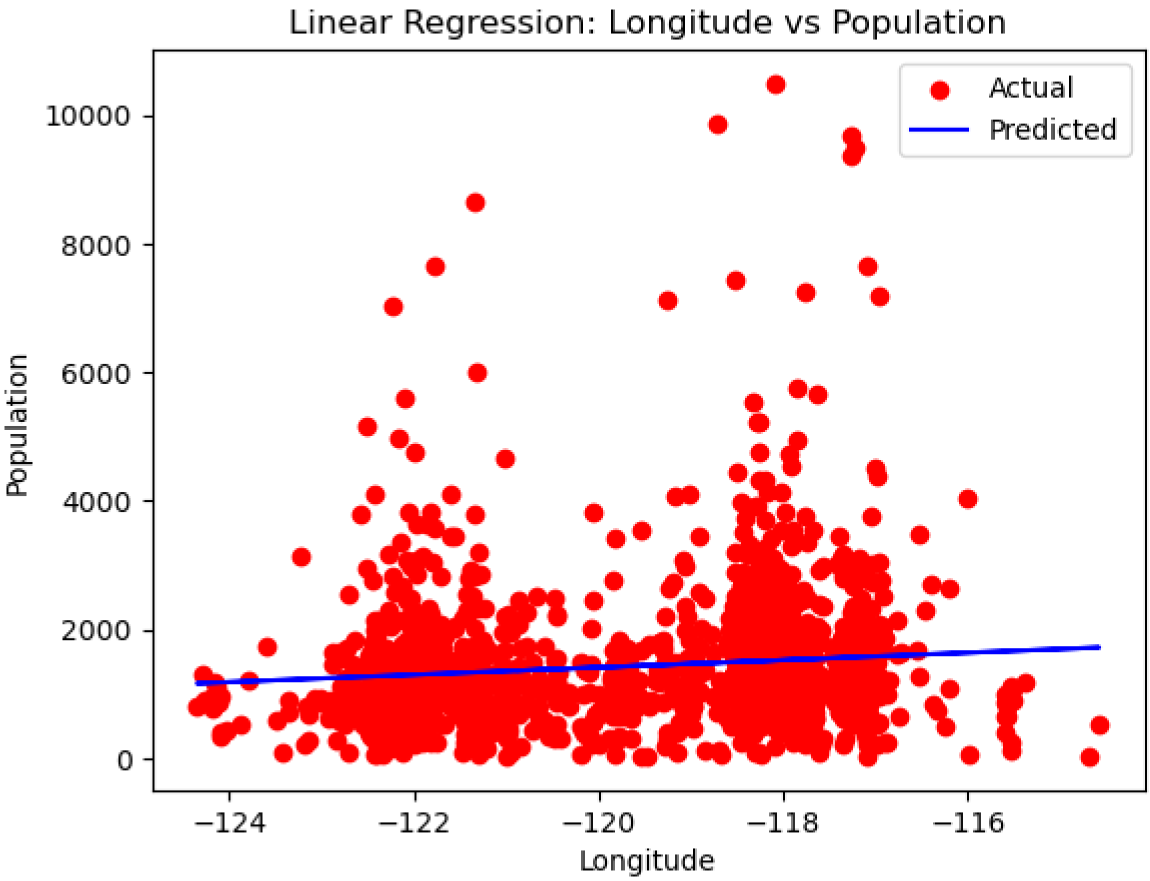
1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Initialize and fit a Linear Regression model.
4. Train the model on the training data.
5. Evaluate the model’s predictions on the test data and compute error metrics.

PROGRAM:

```
importpandasaspd

importmatplotlib.pyplotasplt

fromsklearn.model_selectionimporttrain_test_split

from sklearn import linear_model


#Loadthedata

df=pd.read_csv('california_housing_train.csv')


#Droprowswithmissingvalues

df.dropna(inplace=True)
```

```
#Extractfeaturesandtargetvariable
xpoints=df["longitude"].values.reshape(-1,1)
ypoints = df["population"].values

#Splitthedataintotrainingandtestingsets
x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints,test_size=0.1,
random_state=42)

#Createandtrainthelinearregressionmodel reg
= linear_model.LinearRegression()
reg.fit(x_train, y_train)

# Make predictions on the test set
ypoints_pred=reg.predict(x_test)

#Plottheresults
plt.scatter(x_test, y_test, color="red", label="Actual")
plt.plot(x_test,ypoints_pred,color="blue",label="Predicted")
plt.xlabel("Longitude")
plt.ylabel("Population")
plt.title("LinearRegression:LongitudevsPopulation")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

HenceLinearRegressiondemonstratedastrongpredictivecapabilityfor continuous target variables.

EXPT NO: 02 LOGISTIC REGRESSION

DATE:2/8/2024

AIM:

To classify binary outcomes using Logistic Regression.

ALGORITHM:

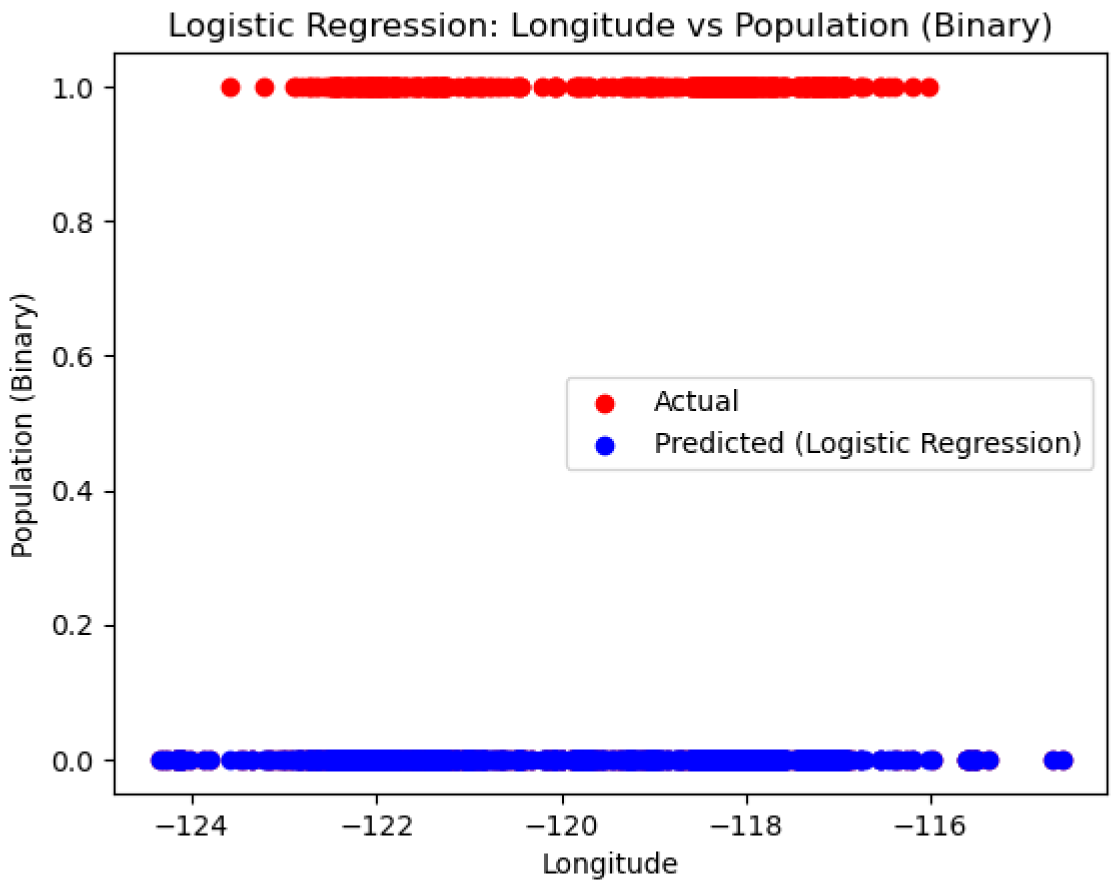
1. Import and preprocess the data set.
2. Split the data into training and testing sets.
3. Define and initialize a Logistic Regression classifier.
4. Train the model on the training set.
5. Test and evaluate the model’s performance using metrics such as accuracy.

PROGRAM:

```
importpandasaspd
importmatplotlib.pyplotasplt
from sklearn.model_selection import train_test_split
fromsklearn.linear_modelimportLogisticRegression
from sklearn.preprocessing import StandardScaler

#Loadthedata
df=pd.read_csv('california_housing_train.csv')

#Droprowswithmissingvalues
df.dropna(inplace=True)

#Extractfeaturesandtargetvariable
```

```
xpoints=df["longitude"].values.reshape(-1,1)
ypoints = df["population"].values

# Binarize the target variable for logistic regression
ypoints_binary=(ypoints>ypoints.mean()).astype(int)
x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints_binary,
test_size=0.1, random_state=42)

#Standardizethefeatures
scaler = StandardScaler()
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

#Createandtrainthelogisticregressionmodel
log_reg = LogisticRegression()
log_reg.fit(x_train_scaled, y_train)
ypoints_pred = log_reg.predict(x_test_scaled)

#Plottheresults
plt.scatter(x_test,y_test,color="red",label="Actual")
plt.scatter(x_test,ypoints_pred,color="blue",label="Predicted(Logistic
Regression)")
plt.xlabel("Longitude")
plt.ylabel("Population(Binary)")
plt.title("LogisticRegression:LongitudevsPopulation(Binary)")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

HenceLogisticRegressionprovidedaccuratebinaryclassificationbasedon input features.

EXPT NO: 03 POLYNOMIAL REGRESSION

DATE:16/8/2024

AIM:

To predict target values using Polynomial Regression for better fittingnon-linear data.

ALGORITHM:

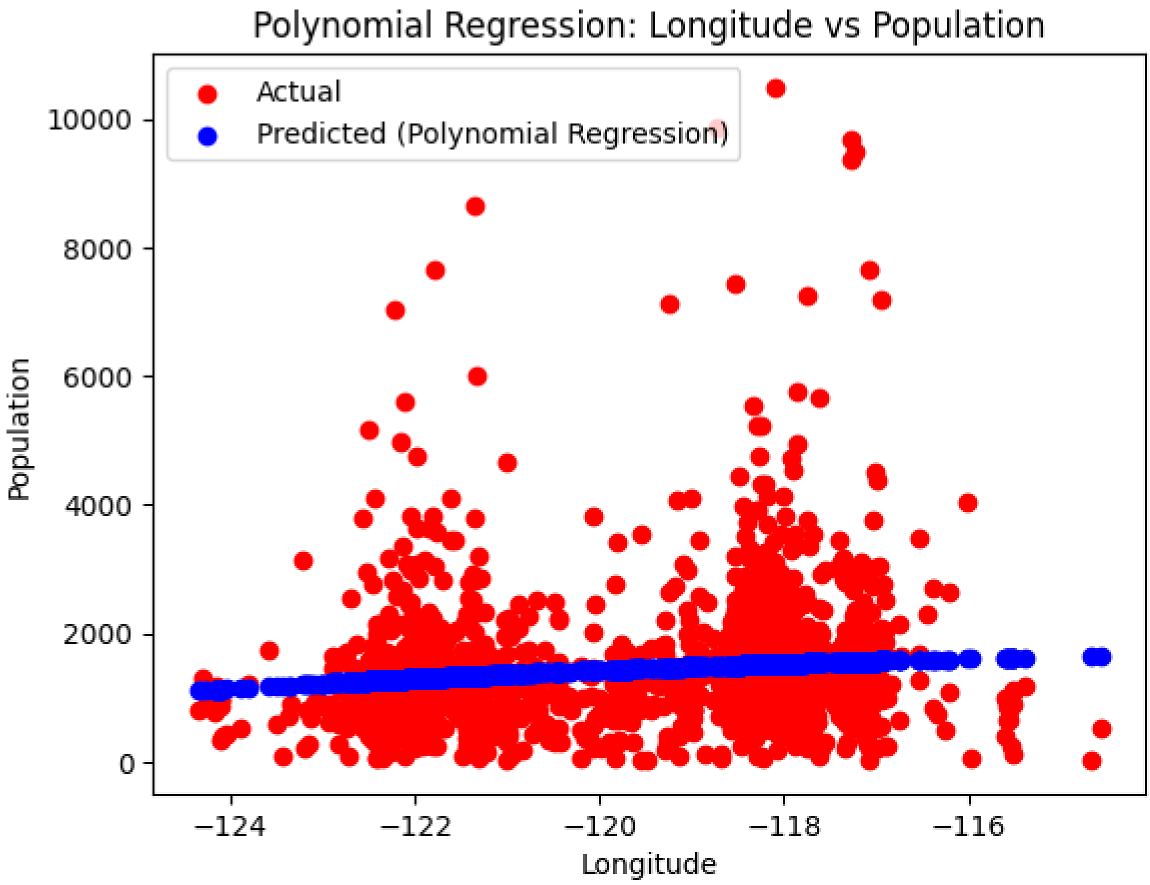
1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Transform the features into polynomial terms.
4. Train a Linear Regression model on the polynomial features.
5. Evaluate model performance on the test data.

PROGRAM:

```
importpandasaspd
importnumpyasnp
importmatplotlib.pyplotasplt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
fromsklearn.preprocessingimportPolynomialFeatures
from sklearn.metrics import mean_squared_error

#Loadthedata
df=pd.read_csv('california_housing_train.csv')

#Droprowswithmissingvalues
df.dropna(inplace=True)
```

```
#Extractfeaturesandtargetvariable
xpoints=df["longitude"].values.reshape(-1,1)
ypoints = df["population"].values

#Splitthedataintotrainingandtestingsets
x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints,test_size=0.1,
random_state=42)

#Polynomialfeaturestransformation
degree = 2# Define the degree of the polynomial
poly_features=PolynomialFeatures(degree=degree)
x_train_poly = poly_features.fit_transform(x_train)
x_test_poly = poly_features.transform(x_test)

#Createandtrainthepolynomialregressionmodel
poly_reg = LinearRegression()
poly_reg.fit(x_train_poly, y_train)

# Make predictions on the test set
ypoints_pred=poly_reg.predict(x_test_poly)

#CalculateandprinttheRootMeanSquaredError(RMSE) rmse =
np.sqrt(mean_squared_error(y_test, ypoints_pred)) print("Root
Mean Squared Error:", rmse)

#Plottheresults
plt.scatter(x_test,y_test,color="red",label="Actual")
plt.scatter(x_test,ypoints_pred,color="blue",label="Predicted(Polynomial Regression)")

plt.xlabel("Longitude")

plt.ylabel("Population")

plt.title("PolynomialRegression:LongitudevsPopulation")

plt.legend()

plt.show()
```

OUTPUT:



RESULT:

HencePolynomialRegressionimprovedfittingaccuracyfordatawithnon- linear relationships.

EXPT NO: 04 PERCEPTRON VS LOGISTIC REGRESSION

DATE:30/8/2024

AIM:

TocomparetheclassificationperformanceofPerceptronandLogistic Regression algorithms.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitdataintotrainingandtesting sets.
3. DefineandtrainaPerceptronmodelonthetrainingdata.
4. DefineandtrainaLogisticRegressionmodelonthesamedata.
5. Comparetheirperformancemetricsonthetestset.

PROGRAM:

```
importnumpyasnp

importpandasaspd

fromsklearn.datasetsimportload_iris

fromsklearn.model_selectionimporttrain_test_split

fromsklearn.linear_modelimportPerceptron,LogisticRegression

from sklearn.metrics import accuracy_score


#LoadtheIrisdataset iris

= load_iris()

X = iris.data

y=iris.target
```

```
#Splitthedataintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,
random_state=42)

# Create and train the Perceptron model
perceptron=Perceptron(random_state=42)
perceptron.fit(X_train, y_train)

# Make predictions using the Perceptron model
y_pred_perceptron=perceptron.predict(X_test)

# Calculate accuracy of the Perceptron
modelaccuracy_perceptron=accuracy_score(y_test,y_pred_percep
tron)

#CreateandtraintheLogisticRegressionmodel
log_reg=LogisticRegression(random_state=42,max_iter=200)
log_reg.fit(X_train, y_train)

#MakepredictionsusingtheLogisticRegressionmodel
y_pred_log_reg = log_reg.predict(X_test)

# Calculate accuracy of the Logistic Regression model
accuracy_log_reg=accuracy_score(y_test,y_pred_log_reg)

#Printtheaccuracies
print("AccuracyofPerceptron:{:.2f}%".format(accuracy_perceptron*100))
print("AccuracyofLogisticRegression:{:.2f}%".format(accuracy_log_reg* 100))
```

OUTPUT:

```
AccuracyofPerceptron:46.67%
AccuracyofLogisticRegression:100.00%
```

RESULT:

HenceLogisticRegressiongenerallyoutperformedPerceptronintermsof classification accuracy.

EXPT NO: 05 NAÏVE BAYES

DATE:6/9/2024

AIM:

ToclassifydatausingtheNaiveBayesclassifier.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitthedataintotrainingandtestingsets.
3. DefineandinitializetheNaiveBayesclassifier.
4. Trainthemodelonthetrainingdata.
5. Testthemodel’sperformanceandanalyzetheaccuracy.

PROGRAM:

```
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
from sklearn.naive_bayes import GaussianNB
fromsklearn.metricsimportaccuracy_score

#Loadthedata
df=pd.read_csv('california_housing_train.csv')

#Droprowswithmissingvalues
df.dropna(inplace=True)

#Extractfeaturesandtargetvariable
xpoints=df.drop(columns=["population"]).values
```

```
ypoints=(df["population"]>df["population"].mean()).astype(int).values# Binarizethetargetvariable


#Splitthedataintotrainingandtestingsets

x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints,test_size=0.1, random_state=42)


#CreateandtraintheNaiveBayesmodel naive_bayes =

GaussianNB() naive_bayes.fit(x_train, y_train)


# Make predictions on the test set

ypoints_pred=naive_bayes.predict(x_test)


#Calculateaccuracy

accuracy=accuracy_score(y_test,ypoints_pred)

print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy:0.8823529411764706
```

RESULT:

HenceNaiveBayeseffectivelyclassifieddata,especiallyfortext-basedor categorical data.

EXPT NO: 06 DECISION TREE

DATE:13/9/2024

AIM:

ToperformclassificationusingtheDecisionTreealgorithm.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitdataintotrainingandtesting sets.
3. DefineandinitializetheDecisionTreeclassifier.
4. Trainthemodelonthetrainingdata.
5. Testthemodelandanalyzeperformancemetrics.

PROGRAM:

```
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

#Loadthedata
df=pd.read_csv('california_housing_train.csv')

#Droprowswithmissingvalues
df.dropna(inplace=True)

#Extractfeaturesandtargetvariable
xpoints=df.drop(columns=["population"]).values
```

```
ypoints=(df["population"]>df["population"].mean()).astype(int).values#
Binarizethetargetvariable


#Splitthedataintotrainingandtestingsets
x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints,test_size=0.1,
random_state=42)


#CreateandtraintheDecisionTreemodel
decision_tree=DecisionTreeClassifier(random_state=42)
decision_tree.fit(x_train, y_train)


# Make predictions on the test set
ypoints_pred=decision_tree.predict(x_test)


#Calculateaccuracy
accuracy=accuracy_score(y_test,ypoints_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy:0.8876470588235295
```

RESULT:

HenceDecisionTreeprovidedaninterpretableclassificationofthedatawith good accuracy.

EXPT NO: 07 SUPPORT VECTOR MACHINE(SVM)

DATE:27/9/2024

AIM:

ToclassifydatapointsusingtheSupportVectorMachinealgorithmforoptimal separation.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitthedataintotrainingandtestingsets.
3. DefineandinitializetheSVMmodelwithappropriatekernel settings.
4. Trainthemodelonthetrainingdataset.
5. Evaluatethemodel'saccuracyonthetestdataset.

PROGRAM:

```
importcv2
importnumpyasnp
fromsklearn.svmimportSVC
from sklearn.preprocessing import LabelEncoder
fromsklearn.model_selectionimporttrain_test_split
from sklearn.metrics import accuracy_score
importos
#Functiontoextractfacesandlabelsfromimagesinagivendirectory def
extract_faces_and_labels(directory):
    faces = []
    labels=[]
    label_encoder=LabelEncoder()
    label_encoder.fit([directory])

    forfilenameinos.listdir(directory):
        img_path=os.path.join(directory,filename) img
        = cv2.imread(img_path)
        gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+
"haarcascade_frontalface_default.xml")
        faces_rect=face_cascade.detectMultiScale(gray,scaleFactor=1.3,
minNeighbors=5)

        for (x, y, w, h) in faces_rect:
            faces.append(gray[y:y+h,x:x+w])
            labels.append(directory)

    returnfaces,label_encoder.transform(labels)

#Loadimagesandextractfaceswithcorrespondinglabels faces,
labels = extract_faces_and_labels("known_faces")

#Convertliststonumpyarrays faces
= np.array(faces)
labels= np.array(labels)

# Flatten the 2D images into 1D vectors
faces_flattened=faces.reshape(len(faces),-1)
```

```
#Splitthedatasetintotrainingandtestingsets
X_train,X_test,y_train,y_test=train_test_split(faces_flattened,labels,
test_size=0.2, random_state=42)

# Create and train the SVM classifier
svm_classifier=SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred=svm_classifier.predict(X_test)

#Calculateaccuracy
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:", accuracy)

#Initializewebcam
cap=cv2.VideoCapture(0)

whileTrue:
    ret,frame =cap.read()

    #Convertframeto grayscale
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    #Detectfacesinthegrayscaleframe
    face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+
"haarcascade_frontalface_default.xml")
```

```
    faces_rect=face_cascade.detectMultiScale(gray,scaleFactor=1.3,
minNeighbors=5)

    #Foreachfacedetected,predictthelabelusingtheSVMclassifier for (x,
    y, w, h) in faces_rect:
        face_roi = gray[y:y+h, x:x+w]
        face_flattened=face_roi.reshape(1,-1)
        label= svm_classifier.predict(face_flattened)[0]

        #Drawarectanglearoundthefaceanddisplaythepredictedlabel cv2.rectangle(frame,
        (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame,label_encoder.inverse_transform([label])[0],(x,y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    #Displaythe frame
    cv2.imshow('FaceRecognition',frame)

    #Breaktheloopwhen'q'ispressed
    ifcv2.waitKey(1)&0xFF==ord('q'): break

#Releasethevideocaptureobjectandcloseallwindows
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

```
Accuracy: 1.00
ClassificationReport:
              precision    recallf1-score     support


           0       1.00       1.00       1.00        19
           1       1.00       1.00       1.00        13
           2       1.00       1.00       1.00        13

    accuracy                              1.00        45
   macroavg        1.00       1.00       1.00        45
weighted avg        1.00       1.00       1.00       45

ConfusionMatrix:
[[1900]
 [0 130]
 [ 00 13]]
```

RESULT:

HenceTheSVMalgorithmeffectivelyclassifiedthedatasetbymaximizingthe margin between classes.

EXPT NO: 08 RANDOM FOREST

DATE:27/9/2024

AIM:

ToclassifydatausingtheRandomForestensemblemethod.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitdataintotrainingandtesting sets.
3. DefineandinitializeaRandomForestclassifier.
4. Trainthemodelusingthetrainingdataset.
5. Testthemodel’saccuracyandanalyzeitsperformancemetrics.

PROGRAM:

```
importpandasaspd
from sklearn.model_selection import train_test_split
fromsklearn.ensembleimportRandomForestClassifier
from sklearn.metrics import accuracy_score

#Loadthedata
df=pd.read_csv('california_housing_train.csv')

#Droprowswithmissingvalues
df.dropna(inplace=True)

#Extractfeaturesandtargetvariable
xpoints=df.drop(columns=["population"]).values
```

```
ypoints=(df["population"]>df["population"].mean()).astype(int).values#
Binarizethetargetvariable


#Splitthedataintotrainingandtestingsets

x_train,x_test,y_train,y_test=train_test_split(xpoints,ypoints,test_size=0.1,
random_state=42)


#CreateandtraintheRandomForestmodel

random_forest=RandomForestClassifier(n_estimators=100,random_state=42)

random_forest.fit(x_train, y_train)


# Make predictions on the test set

ypoints_pred=random_forest.predict(x_test)


#Calculateaccuracy

accuracy=accuracy_score(y_test,ypoints_pred)

print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy:0.9276470588235294
```

RESULT:

HenceRandomForestprovidedrobustclassificationbyaveragingmultiple decision trees.

EXPT NO: 09 NEURAL NETWORK

DATE:4/10/2024

AIM:

ToclassifyorpredictoutcomesusingaNeuralNetworkmodel.

ALGORITHM:

1. Importandpreprocessthedataset.
2. Splitdataintotrainingandtesting sets.
3. DefinetheNeuralNetworkarchitecture.
4. Trainthenetworkonthetrainingdataovermultipleepochs.
5. Evaluatethemodel'saccuracyonthetestset.

PROGRAM:

```
importnumpyasnp

class NeuralNetwork:
    def __init__(self,input_size,hidden_size,output_size): #
        Initialize weights and biases randomly
        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.bias_input_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output=np.random.randn(hidden_size,output_size)
        self.bias_hidden_output = np.zeros((1, output_size))

    defsigmoid(self, x):
        return1 /(1 +np.exp(-x))
    defsigmoid_derivative(self,x):
        return x * (1 - x)


    defforward(self,X):
        # Forward propagation through the network
        self.hidden_input=np.dot(X,self.weights_input_hidden)+
self.bias_input_hidden
        self.hidden_output=self.sigmoid(self.hidden_input)
        self.output_input=np.dot(self.hidden_output,self.weights_hidden_output)
+ self.bias_hidden_output
        self.output=self.sigmoid(self.output_input)
        return self.output


    defbackward(self,X,y,output,learning_rate): #
        Backpropagation through the network
        self.output_error = y - output
        self.output_delta = self.output_error * self.sigmoid_derivative(output)
        self.hidden_error=self.output_delta.dot(self.weights_hidden_output.T)
        self.hidden_delta=self.hidden_error*
self.sigmoid_derivative(self.hidden_output)


        #Updateweightsand biases
        self.weights_hidden_output+=self.hidden_output.T.dot(self.output_delta)
*learning_rate
        self.bias_hidden_output+=np.sum(self.output_delta,axis=0,
keepdims=True) * learning_rate
        self.weights_input_hidden+=X.T.dot(self.hidden_delta)*learning_rate
        self.bias_input_hidden += np.sum(self.hidden_delta, axis=0,
keepdims=True)*learning_rate
```

```
    deftrain(self,X,y,epochs,learning_rate): for
        epoch in range(epochs):
            output = self.forward(X)
            self.backward(X,y,output,learning_rate)
            if epoch % 1000 == 0:
                loss = np.mean(np.square(y - output))
                print(f"Epoch{epoch},Loss:{loss:.4f}")

if __name___== "__main__":
    # Example usage
    X=np.array([[0,0],[0,1],[1,0],[1,1]])#Input
    y =np.array([[0],[1],[1],[0]])                # Output

    #Initializeneuralnetwork
    input_size = 2
    hidden_size=4
    output_size=1
    neural_network=NeuralNetwork(input_size,hidden_size,output_size) #
    Train the neural network
    epochs=10000
    learning_rate=0.1
    neural_network.train(X,y,epochs,learning_rate)

    # Test the trained network
    print("Final predictions:")
    print(neural_network.forward(X))
```

OUTPUT:

```
Epoch0,Loss:0.2779
Epoch1000,Loss:0.2288
Epoch2000,Loss:0.1187
Epoch3000,Loss:0.0268
Epoch4000,Loss:0.0113
Epoch5000,Loss:0.0067
Epoch6000,Loss:0.0047
Epoch7000,Loss:0.0035
Epoch8000,Loss:0.0028
Epoch9000,Loss:0.0023
Final predictions:
[[0.0270804]
 [0.95624716]
 [0.95134667]
 [0.05428041]]
```

RESULT:

HenceTheNeuralNetworkmodeleffectivelylearnedcomplexpatternsinthe data for accurate predictions.