ITA06

MACHINE LEARNING

LAB RECORD

DEPARTMENT OF CS

SAVEETHA SCHOOL OF ENGINEERING

SAVEETHA UNIVERSITY

CHENNAI 602 105

INDEX

S.NO.	DATE	PROGRAM	SIGNATURE
		UNIT - 1	
1.		FIND-S ALGORITHM	
2.		CANDIDATE-ELIMINATION ALGORITHM	
3.		ID3 ALGORITHM	
		UNIT - 2	
4.		BACKPROPAGATION ALGORITHM	
		UNIT - 3	
5.		NAÏVE BAYES	
6.		LOGISTIC REGRESSION	
7.		LINEAR REGRESSION	
8.		LINEAR & POLYNOMIAL	
9.		EM ALGORITHM	
		UNIT - 4	
10.		K-NEAREST NEIGHBOR ALGORITHM	
		UNIT - 5	
11.		WRITE A PROGRAM FOR THE TASK OF CREDIT SCORE CLASSIFICATION	

	DATE	**PROGRAM**	**SIGNATURE**
12.		**IRIS FLOWER CLASSIFICATION USING KNN**	
13.		**CAR PRICE PREDICTION MODEL USING PYTHON**	
14.		**HOUSE PRICE PREDICTION USING MACHINE LEARNING IN PYTHON**	
15.		**NAÏVE IRIS CLASSIFICATION**	
16.		**COMPARISON OF CLASSIFICATION ALGORITHMS**	
17.		**MOBILE PRICE CLASSIFICATION USING PYTHON**	
18.		**PERCEPTRON IRIS CLASSIFICATION**	
19.		**IMPLEMENTATION OF NAIVE BAYES IN PYTHON – MACHINE LEARNING**	
20.		**FUTURE SALES PREDICTION USING PYTHON**	

Pgm.No.: 1 **FIND-S ALGORITHM**
Date:

Aim:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Procedure:

FIND-S Algorithm

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training
      instance x For each attribute
      constraint ai in h
         If the constraint ai is satisfied by x
         Then do nothing
         Else replace ai in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program

```
import csva

= []


with open('enjoysport.csv', 'r') as csvfile:for row in
        csv.reader(csvfile):

                a.append(row)
        print(a)
print("\n The total number of training instances are : ",len(a))num_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")hypothesis =
['0']*num_attribute print(hypothesis)


for i in range(0, len(a)):
        if a[i][num_attribute] == 'yes':
                for j in range(0, num_attribute):
                        if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:hypothesis[j] = a[i][j]
                        else:
                                hypothesis[j] = '?'
        print("\n The hypothesis for the training instance {} is :
\n" .format(i+1),hypothesis)


print("\n The Maximally specific hypothesis for the traininginstance is ")
print(hypothesis)
```

Output:

```
The Given Training Data Set
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']The total number of
training instances are :   4

The initial hypothesis is :
  ['0', '0', '0', '0', '0', '0']
```

```
The hypothesis for the training instance 1 is : ['sunny', 'warm', 'normal',
  'strong', 'warm', 'same']
The hypothesis for the training instance 2 is : ['sunny', 'warm', '?', 'strong',
  'warm', 'same']
The hypothesis for the training instance 3 is : ['sunny', 'warm', '?', 'strong',
  'warm', 'same']
The hypothesis for the training instance 4 is :['sunny', 'warm',
  '?', 'strong', '?', '?']
The Maximally specific hypothesis for the training instance is
  ['sunny', 'warm', '?', 'strong', '?', '?']
```

Result:

Pgm.No.: 2 **CANDIDATE-ELIMINATION ALGORITHM**
Date:

Aim:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

The CANDIDATE-ELIMINTION algorithm computes the version space containing allhypotheses from H that are consistent with an observed sequence of training examples.

Procedure

Initialize G to the set of maximally general hypotheses in HInitialize S to the set of maximally specific hypotheses in HFor each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

CANDIDATE- ELIMINTION algorithm using version spaces

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program

```
  import numpy as np import
  pandas as pd

  data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))concepts =
  np.array(data.iloc[:,0:-1])

  print(concepts)

  target = np.array(data.iloc[:,-1])print(target)


  def learn(concepts, target):

          specific_h = concepts[0].copy()

print("initialization of specific_h and general_h")

print(specific_h)

general_h = [["?" for i in range(len(specific_h))] for i inrange(len(specific_h))]

print(general_h)

          for i, h in enumerate(concepts):

          if target[i] == "yes":

                      for x in range(len(specific_h)):

                      if h[x]!= specific_h[x]:

                                  specific_h[x] ='?'
                                  general_h[x][x] ='?'

                print(specific_h)

                print(specific_h)

                if target[i] == "no":

                      for x in range(len(specific_h)):if h[x]!=
                            specific_h[x]:
```

```
                        general_h[x][x] = specific_h[x]else:
                        general_h[x][x] = '?'
            print(" steps of Candidate Elimination Algorithm",i+1)print(specific_h)
            print(general_h)
        indices = [i for i, val in enumerate(general_h) if val ==['?', '?', '?', '?', '?', '?']]
        for i in indices:
            general_h.remove(['?', '?', '?', '?', '?', '?'])
        return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Output:

```
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']


Final General_h:
[['sunny', '?', '?', '?', '?', '?'],
 ['?', 'warm', '?', '?', '?', '?']]
```

Result:

Pgm.No.: 3 **ID3 Algorithm**

Date:

Aim:

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Procedure

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to bepredicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common valueof Target_attribute in Examples

- Otherwise Begin
  - A ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = v_i
    - Let *Examples* v_i, be the subset of Examples that have value v_i for *A*
    - If *Examples* v_i , is empty
      - Then below this new branch add a leaf node with label = most commonvalue of Target_attribute in Examples
      - Else below this new branch add the subtree ID3(*Examples* v_i, Targe_tattribute, Attributes {A}))
- End
- Return Root

ENTROPY:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where, *p+* is the proportion of positive examples in S

p- is the proportion of negative examples in S.

INFORMATION GAIN:

- ***Information gain,*** is the expected reduction in entropy caused by partitioning theexamples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examplesS, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

<u>Training Dataset:</u>

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Test Dataset:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Program

```
import math
import csv


def load_csv(filename):  lines=csv.reader(open(filename,"r"));

          dataset = list(lines) headers =
          dataset.pop(0)return
          dataset,headers



class Node:

          def___init__(self,attribute):
                    self.attribute=attribute
                    self.children=[] self.answer=""


def subtables(data,col,delete):  dic={}

          coldata=[row[col] for row in data]
          attr=list(set(coldata))


          counts=[0]*len(attr)
          r=len(data) c=len(data[0])

          for x in range(len(attr)):for y in
                    range(r):

                              if data[y][col]==attr[x]:
                                        counts[x]+=1
```

```
        for x in range(len(attr)):

            dic[attr[x]]=[[0 for i in range(c)] for j inrange(counts[x])]

            pos=0

            for y in range(r):

                if data[y][col]==attr[x]:if delete:

                        del data[y][col]
                    dic[attr[x]][pos]=data[y] pos+=1

        return attr,dic

def entropy(S):     attr=list(set(S))

        if len(attr)==1:return
            0


        counts=[0,0]

        for i in range(2):

            counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)


        sums=0

        for cnt in counts:

            sums+=-1*cnt*math.log(cnt,2) return
        sums


def compute_gain(data,col):     attr,dic = subtables(data,col,delete=False)


        total_size=len(data)
        entropies=[0]*len(attr)
        ratio=[0]*len(attr)


        total_entropy=entropy([row[-1] for row in data])for x in
        range(len(attr)):
                ratio[x]=len(dic[attr[x]])/(total_size*1.0)
                entropies[x]=entropy([row[-1] for row in
dic[attr[x]]])
                total_entropy-=ratio[x]*entropies[x] return
        total_entropy


def build_tree(data,features):   lastcol=[row[-1] for row in data]
        if(len(set(lastcol)))==1:
                node=Node("")
                node.answer=lastcol[0]return
                node


        n=len(data[0])-1
        gains=[0]*n
        for col in range(n): gains[col]=compute_gain(data,col)
        split=gains.index(max(gains))
        node=Node(features[split])
        fea = features[:split]+features[split+1:] attr,dic=subtables(data,split,delete=True)


        for x in range(len(attr)): child=build_tree(dic[attr[x]],fea)
                node.children.append((attr[x],child))
        return node


def print_tree(node,level): if node.answer!="":
                print("  "*level,node.answer)return


        print("           "*level,node.attribute) for
        value,n in node.children:
                print("  "*(level+1),value)
                print_tree(n,level+2)



def classify(node,x_test,features):   if node.answer!="":
```

```
            print(node.answer)
            return

        pos=features.index(node.attribute) for value, n
        in node.children:

            if x_test[pos]==value: classify(n,x_test,features)


'''Main program'''  dataset,features=load_csv("data3.csv")
node1=build_tree(dataset,features)


print("The decision tree for the dataset using ID3 algorithmis")

print_tree(node1,0)
testdata,features=load_csv("data3_test.csv") for xtest in
testdata:

        print("The test instance:",xtest)

        print("The label for test instance:",end="     ")
        classify(node1,xtest,features)
```

Output:

```
Decision Tree is:
outlook
        overcast ->  ['yes']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot',
'humidity': 'normal', 'wind': 'strong'}  is: ['yes']
```

Pgm.No.: 4 **BACKPROPAGATION ALGORITHM**

Date:

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Procedure

BACKPROPAGATION (training_example, ɳ, nin, nout, nhidden)

Each training example is a pair of the form ($\vec{x}$ →,

t→), where (x→) is the vector of network

input values, (t→) and is the vector of target network output values.
ɳ is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units.
The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji

- Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

- For each ($\vec{x}$→,

t→), in training examples, Do

Propagate the input forward through the network:
1. Input the instance $\vec{x}$→, to the network and compute the output ou of every unit u in the network.
2. Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Program

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
   return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
```

```
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
   #Forward Propogation
   hinp1=np.dot(X,wh)
   hinp=hinp1 + bh
   hlayer_act = sigmoid(hinp)
   outinp1=np.dot(hlayer_act,wout)
   outinp= outinp1+bout
   output = sigmoid(outinp)

   #Backpropagation
   EO = y-output
   outgrad = derivatives_sigmoid(output)
   d_output = EO * outgrad
   EH = d_output.dot(wout.T)
   hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to
error
   d_hiddenlayer = EH * hiddengrad

   wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and
currentlayerop
   wh += X.T.dot(d_hiddenlayer) *lr

   print ("-----------Epoch-", i+1, "Starts----------")
   print("Input: \n" + str(X))
   print("Actual Output: \n" + str(y))
   print("Predicted Output: \n" ,output)
   print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
```

```
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.79623473]
 [0.78886338]
 [0.7952609 ]]
-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.79781876]
 [0.79037225]
 [0.79683561]]
-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.79935983]
 [0.79184099]
 [0.79836775]]
-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80085967]
 [0.79327116]
 [0.79985899]]
-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80231989]
 [0.79466427]
 [0.80131096]]
-----------Epoch- 5 Ends----------

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80231989]
 [0.79466427]
 [0.80131096]]
```

Result:

Exp.no: 5 **NA¯VE BAYESIAN CLASSIFIER**
Date:

Aim Assuming a set of documents that need to be classified, use the naÃ¯ve Bayesian Classifier model to perform this task. Calculate the accuracy, precision, and recall for your data set.

Procedure

The Naïve Bayesian classifier is a popular machine learning algorithm for classification tasks. It is based on the Bayes theorem and assumes that the features are independent of each other.The breast cancer dataset is a popular dataset used for classification tasks. It contains various features of breast cancer patients, such as the size of the tumor, the age of the patient, etc., and the class label, which indicates whether the tumor is benign or malignant.
To apply the Naïve Bayesian classifier to the breast cancer dataset, you can follow these steps
Load the dataset: You can load the breast cancer dataset using any programming language and libraries of your choice, such as Python and scikit-learn.
Preprocess the data: You may need to preprocess the data by handling missing values, scaling the data, and encoding categorical variables.
Split the dataset: Split the dataset into training and testing sets. You can use 75% of the data for training and 25% for testing.
Train the Naïve Bayesian classifier: Train the classifier on the training data using the Gaussian Naïve Bayesian algorithm, which assumes that the features follow a Gaussian distribution.
Make predictions: Use the trained classifier to make predictions on the testing data.
Evaluate the performance: Evaluate the performance of the classifier using metrics such as accuracy, precision, recall, and F1-score.

Program

```
import csv
import
random
import math


def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));dataset =
        list(lines)

        for i in range(len(dataset)):

          #converting strings into numbers for processing

                dataset[i] = [float(x) for x in dataset[i]]


        return dataset


def splitdataset(dataset, splitratio):

      #67% training size

        trainsize = int(len(dataset) * splitratio);

        trainset = []

        copy = list(dataset);

        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele fortraining data

                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))

        return [trainset, copy]


def separatebyclass(dataset):

        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are#the instances belonging to each
class
        for i in range(len(dataset)):vector =
                dataset[i]

                if (vector[-1] not in separated):
                        separated[vector[-1]] = []

                separated[vector[-1]].append(vector)return
        separated


def mean(numbers):
        return sum(numbers)/float(len(numbers))


    def stdev(numbers):

        avg = mean(numbers)

        variance = sum([pow(x-avg,2) for x in
    numbers])/float(len(numbers)-1)

        return math.sqrt(variance)

    def summarize(dataset): #creates a dictionary of classes
        summaries       =     [(mean(attribute),              stdev(attribute))
        for

    attribute in zip(*dataset)];

        del summaries[-1] #excluding labels +ve or -vereturn
        summaries


    def summarizebyclass(dataset):

        separated = separatebyclass(dataset);
       #print(separated)

        summaries = {}

        for classvalue, instances in separated.items():
#for key,value in dic.items()

#summaries is a dic of tuples(mean,std) for each class value
            summaries[classvalue] = summarize(instances)#summarize
    is used to cal to mean and std

        return summaries


    def calculateprobability(x, mean, stdev): exponent = math.exp(-
        (math.pow(x-mean,2)/

    (2*math.pow(stdev,2))))

        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent


    def calculateclassprobabilities(summaries, inputvector):

    # probabilities contains the all prob of all class of test dataprobabilities = {}
        for classvalue, classsummaries in summaries.items(): #class and attribute
    information as mean and sd

            probabilities[classvalue] = 1
            for i in range(len(classsummaries)):
                mean, stdev = classsummaries[i] #take mean andsd of every
    attribute for class 0 and 1 seperaely
                x = inputvector[i] #testvector's first attribute
                probabilities[classvalue] *=
    calculateprobability(x, mean, stdev);#use normal distreturn probabilities


    def predict(summaries, inputvector): #training and test datais passed
        probabilities = calculateclassprobabilities(summaries,inputvector)
        bestLabel, bestProb = None, -1
        for classvalue, probability in probabilities.items():#assigns that class
    which has the highest prob
            if bestLabel is None or probability > bestProb:bestProb =
                probability
                bestLabel = classvalue
        return bestLabel
    def getpredictions(summaries, testset):predictions =
        []
        for i in range(len(testset)):
            result = predict(summaries, testset[i])
            predictions.append(result)
        return predictions


    def getaccuracy(testset, predictions):correct = 0
        for i in range(len(testset)):
            if testset[i][-1] == predictions[i]:correct += 1
        return (correct/float(len(testset))) * 100.0
```

```
def main():

        filename = 'naivedata.csv'
        splitratio = 0.67

        dataset = loadcsv(filename);


        trainingset, testset = splitdataset(dataset, splitratio)print('Split {0} rows into
        train={1} and test={2}

rows'.format(len(dataset), len(trainingset), len(testset)))# prepare model

        summaries = summarizebyclass(trainingset);
        #print(summaries)

       # test model

        predictions = getpredictions(summaries, testset) #find thepredictions of test data
with the training data

        accuracy = getaccuracy(testset, predictions)print('Accuracy
        of the classifier is :

{0}%'.format(accuracy)) main()
```

Output:

```
Split 768 rows into train=514 and test=254

rows Accuracy of the classifier is :

71.65354330708661
```

Result:

Exp.no: 6 LINEAR REGRESSION (LR)
Date:
Aim Write a program to implement Linear Regression (LR) algorithm in python

Procedure:

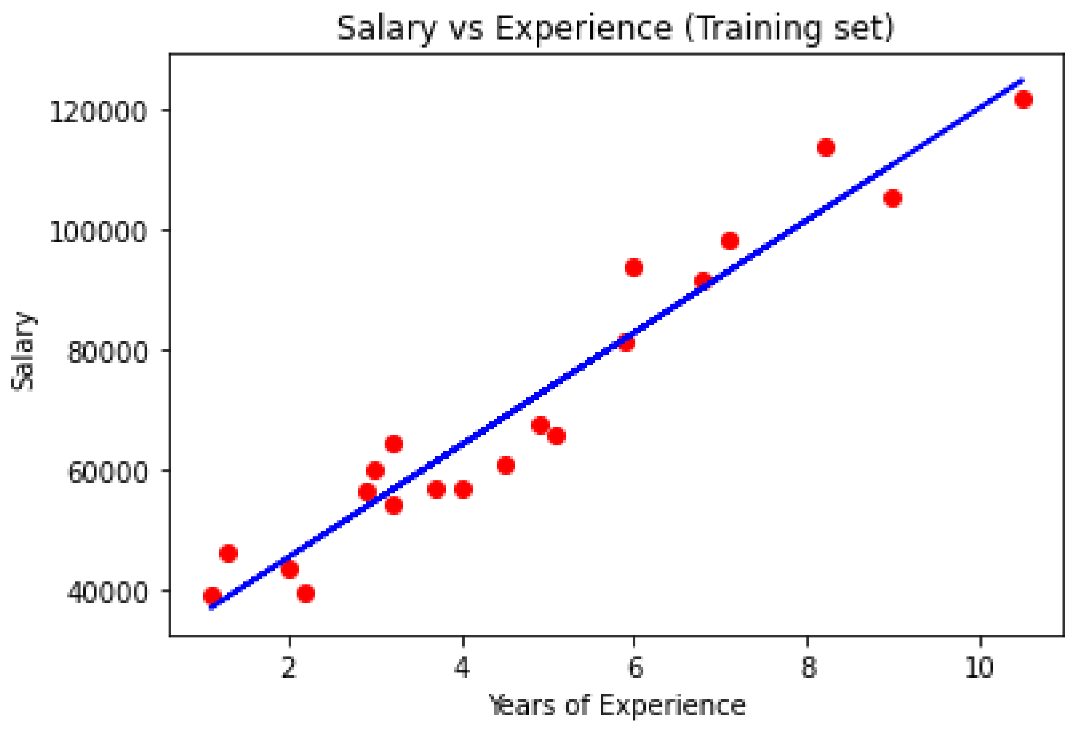
Linear regression is a popular machine learning algorithm used for predicting a continuous target variable. It works by finding the best-fit line that describes the relationship between the input features and the target variable.

The salary dataset is a popular dataset used for regression tasks. It contains various features such as years of experience, education level, job title, etc., and the target variable, which is the salary of the employee.

Program

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

dataset.head()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)


from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

y_pred = regressor.predict(X_test)
pd.DataFrame(data={'Actuals': y_test, 'Predictions': y_pred})


#Visualising the Training set results Here scatter plot is used to visualize the results.

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Output:



Result:

ExpNo:7 LINEAR AND POLYNOMIAL REGRESSION
Date:

Aim: Implementation Of Linear And Polynomial Regression In Python

Procedure:

Linear and Polynomial Regression are two widely used algorithms for predicting a continuous target variable based on one or more input features.

Linear Regression finds the line of best fit that describes the relationship between the input feature and the target variable. It is a simple algorithm that works well when the relationship between the input feature and target variable is linear.

Polynomial Regression, on the other hand, is a more complex algorithm that can capture non-linear relationships between the input features and target variable. It works by adding polynomial terms to the linear regression equation, allowing it to fit more complex curves.

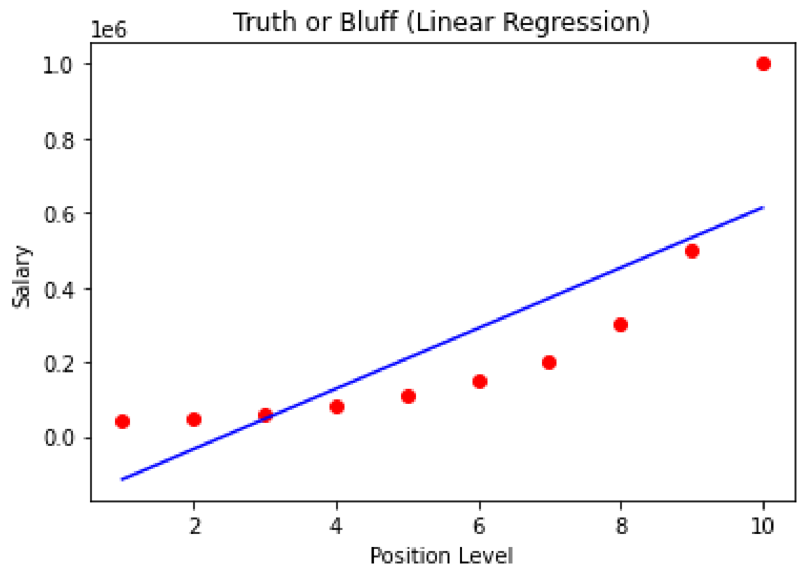
In the given code, a dataset containing information about the position and salary of employees is used to train both Linear and Polynomial Regression models. The Linear Regression model is trained using the entire dataset, while the Polynomial Regression model is trained using the transformed input feature matrix created by adding polynomial terms up to degree 4.

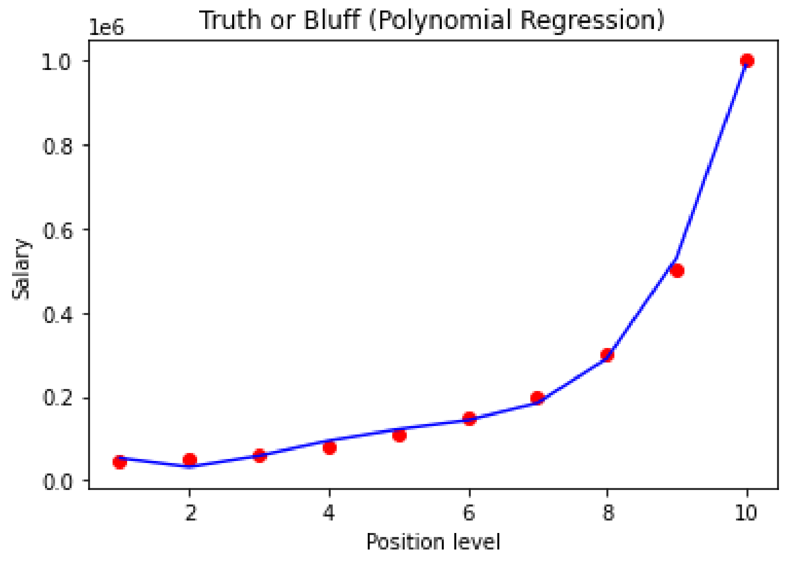
Program

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('Position_Salaries.csv')

X = dataset.iloc[:, 1:-1].values

y = dataset.iloc[:, -1].values

""

Training the Linear Regression model on the Whole dataset

A Linear regression algorithm is used to create a model.

 A LinearRegression function is imported from sklearn.linear_model library.

 ""
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

#Linear Regression classifier model
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
"""
Training the Polynomial Regression model on the Whole dataset
A polynomial regression algorithm is used to create a model.
"""
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
#Polynomial Regression classifier model
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
"""
Visualising the Linear Regression results
Here scatter plot is used to visualize the results. The title of the plot is set to Truth or Bluff
(Linear Regression), xlabel is set to Position Level , and ylabel is set to Salary.
"""
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
#Visualising the Polynomial Regression results
"""
The title of the plot is set to Truth or Bluff (Polynomial Regression), xlabel is set to Position
level,
 and ylabel is set to Salary.
"""

plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Output:





Result:

ExpNo:8 **LOGISTIC REGRESSION (LR)**
Date:

Aim:Write a program to implement Logistic Regression (LR) algorithm in python

Procedure:

Logistic Regression is a popular classification algorithm used in machine learning. It is used to predict the categorical dependent variable based on one or more independent variables. The implementation of Logistic Regression in Python involves importing the required libraries, importing the dataset, dividing the dataset into concepts and targets, splitting the dataset into training and testing sets, feature scaling, training the Logistic Regression (LR) classification model on the training set, and evaluating the model using evaluation metrics such as confusion matrix and accuracy.

In the given implementation, the required libraries are imported at the beginning, followed by reading the breast cancer dataset using pandas. The dataset is divided into concepts and targets, where the concepts are stored in X and targets in y. The dataset is then split into training and testing sets using the train_test_split function from the sklearn.model_selection library. Feature scaling is done using the StandardScaler technique from the sklearn.preprocessing library.

The Logistic Regression (LR) classifier algorithm is used to create a model on the training set with hyperparameters such as random_state set to 0. The classifier is fitted to the training set using the fit function. Evaluation metrics such as confusion matrix and accuracy are then used to evaluate the performance of the model on the test set. The confusion matrix is calculated using the confusion_matrix function from the sklearn.metrics library, and the accuracy is calculated using the accuracy_score function from the same library.

Program

```
"""

Implementation of Logistic Regression (LR) in Python – Machine Learning

Importing the libraries

"""
import numpy as np
import pandas as pd
```

```
#"Importing the dataset
""""
After importing the necessary libraries, next, we import or read the dataset.
 Click here to download the breast cancer dataset used in this implementation.
 The breast cancer dataset has the following features:
 Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape,
Marginal Adhesion,
 Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitosis, Class.
""""

# divide the dataset into concepts and targets. Store the concepts into X and targets into y.
dataset = pd.read_csv("D:/GEO/BE COURSES/2022
dec/LAB/DATASET/breastcancer.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

#Splitting the dataset into the Training set and Test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 2)

#Feature Scaling
""""
Feature scaling is the process of converting the data into a given range. In this case, the
standard scalar technique is used.
from sklearn.preprocessing import StandardScaler
""""
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
""""
Training the Logistic Regression (LR) Classification model on the Training set
Once the dataset is scaled, next, the Logistic Regression (LR) classifier algorithm is used to
create a model.
The hyperparameters such as random_state to 0 respectively.
 The remaining hyperparameters Logistic Regression (LR) are set to default values.
 """"
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
#Logistic Regression (LR) classifier model
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                  warm_start=False)
#Display the results (confusion matrix and accuracy)
"""
Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model
built using a decision tree classifier.
"""
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy Score:confusion matrix')
accuracy_score(y_test, y_pred)
```

Output:

```
[[117   8]
 [  6  74]]
Accuracy Score:confusion matrix
```

Result:

ExpNo:9 EXPECTATION & MAXIMIZATION ALGORITHM
Date:

Aim

Python Program to Implement Estimation & MAximization Algorithm
Program:
Procedure:

The EM (Expectation-Maximization) algorithm is a statistical algorithm used to estimate the parameters of a statistical model with hidden variables. It is an iterative method that alternates between two steps: the E-step (Expectation step) and the M-step (Maximization step).

In the E-step, the algorithm calculates the expected values of the latent variables given the observed data and the current estimate of the model parameters. In the M-step, the algorithm updates the model parameters to maximize the likelihood of the observed data, based on the expected values of the latent variables computed in the E-step.

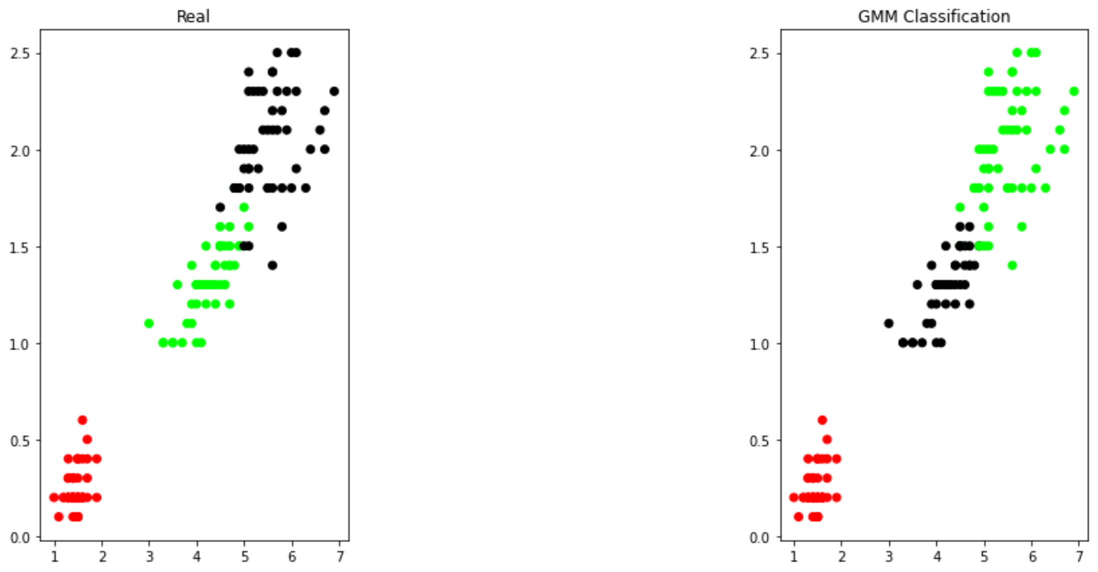
In machine learning, the EM algorithm is often used for clustering, such as in Gaussian mixture models, where each cluster is modeled by a Gaussian distribution. The EM algorithm is used to estimate the parameters of the Gaussian mixture model, such as the means, covariances, and mixture weights, given the observed data. In machine learning, the EM algorithm is often used for clustering, such as in Gaussian mixture models, where each cluster is modeled by a Gaussian distribution. The EM algorithm is used to estimate the parameters of the Gaussian mixture model, such as the means, covariances, and mixture weights, given the observed data.
The EM algorithm can also be used for other machine learning tasks, such as training hidden Markov models, factor analysis, and Bayesian networks.

Program

```
"""
Python Program to Implement  Estimation & MAximization Algorithm
"""
#from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("D:/GEO/BE COURSES/2022 dec/LAB/IRIS.csv")
```

```
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.petal_length
,X.petal_width,c=colormap[y])


# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.petal_length,X.petal_width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
#print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm)
```

Output:



Result:

Exp.No:10 **K-NEAREST NEIGHBOR ALGORITHM**

Date:

Aim: Write a program for Implementation of K-Nearest Neighbors (K-NN) in Python

Procedure

K-Nearest Neighbors (K-NN) is a popular machine learning algorithm for classification and regression tasks. It works by finding the K data points in the training set that are closest to a given test point and then making a prediction based on the labels of those K neighbors. Here's an example Python program that implements K-NN:

1. Import the necessary libraries: Begin by importing the required libraries, such as NumPy and scikit-learn, which provide tools for data manipulation and machine learning algorithms
2. Prepare the dataset: Load and preprocess the dataset, ensuring it is in a format suitable for the algorithm. This may involve converting categorical variables to numerical representations or normalizing features.
3. Split the dataset: Divide the dataset into training and testing sets. The training set will be used to train the KNN model, and the testing set will evaluate its performance.
4. Create and train the KNN model: Instantiate the KNN classifier and specify the desired value for K. Then, fit the model to the training data.
5. Make predictions: Use the trained model to make predictions on the testing dataset.
6. Evaluate the model: Assess the performance of the model by comparing the predicted labels with the actual labels from the testing set. Common evaluation metrics include accuracy, precision, recall, and F1-score.

Program

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix

from sklearn import datasets


iris=datasets.load_iris()


x = iris.data
y = iris.target
```

```
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
Confusion Matrix
[[16  0  0]
 [ 0 14  1]
 [ 0  1 13]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.93      0.93      0.93        15
           2       0.93      0.93      0.93        14

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Result:

Pgm.No.: 11 **Credit Scores prediction**
Date:

Aim:**Credit Scores prediction using Machine Learning in Python**

Procedure

There are three credit scores that banks and credit card companies use to label their customers: Good, Standard, Poor A person with a good credit score will get loans from any bank and financial institution. Write a program for the task of Credit Score Classification, we need a labelled dataset with credit scores.
Credit Score Classification

There are three credit scores that banks and credit card companies use to label their customers:

Good

Standard

Poor

A person with a good credit score will get loans from any bank and financial institution. For the task of Credit Score Classification, we need a labelled dataset with credit scores.

PROCEDURE

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""

Credit Score Classification
There are three credit scores that banks and credit card companies use to label their
customers:

Good
```

Standard
Poor
A person with a good credit score will get loans from any bank and financial institution. For the task of Credit Score Classification, we need a labelled dataset with credit scores.

```
"""

import pandas as pd
import numpy as np

import plotly.express as px

import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"

import plotly.io as io

io.renderers.default='browser'
data = pd.read_csv("D:/GEO/BE COURSES/2022 dec/LAB\DATASET/Credit-Score-Data/Credit Score Data/CREDITSCORE.csv")
print(data.head())
print(data.info())
#the dataset has any null values or not:
print(data.isnull().sum())
#The dataset doesn’t have any null values. As this dataset is labelled, let’s have a look at the Credit_Score column values:
data["Credit_Score"].value_counts()
data.shape
data.describe()
#Data Exploration
"""
```

The dataset has many features that can train a Machine Learning model for credit score classification.
Let’s explore all the features one by one.

I will start by exploring the occupation feature to know if the occupation of the person affects credit scores:

```
"""
fig = px.box(data,
        x="Occupation",
        y="Credit_Score",
        color="Credit_Score",#y
        title="Credit Scores Based on Occupation",
        color_discrete_map={'Poor':'red',
                    'Standard':'yellow',
                    'Good':'green'})
```

```
fig.show()
"""
There’s not much difference in the credit scores of all occupations mentioned in the data.
Now let’s explore
 whether the Annual Income of the person impacts your credit scores or not:
"""
fig = px.box(data,
             x="Credit_Score",
             y="Annual_Income",
             color="Credit_Score",
             title="Credit Scores Based on Annual Income",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()

"""
 let’s explore whether the monthly in-hand salary impacts credit scores or not:
"""
fig = px.box(data,
             x="Credit_Score",
             y="Monthly_Inhand_Salary",
             color="Credit_Score",
             title="Credit Scores Based on Monthly Inhand Salary",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()

fig = px.box(data,
             x="Credit_Score",
             y="Num_Bank_Accounts",
             color="Credit_Score",
             title="Credit Scores Based on Number of Bank Accounts",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()
# impact on credit scores based on the number of credit cards you have:
fig = px.box(data,
             x="Credit_Score",
             y="Num_Credit_Card",
            color="Credit_Score",
            title="Credit Scores Based on Number of Credit cards",
            color_discrete_map={'Poor':'red',
                              'Standard':'yellow',
                              'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()

fig = px.box(data,
            x="Credit_Score",
            y="Interest_Rate",
            color="Credit_Score",
            title="Credit Scores Based on the Average Interest rates",
            color_discrete_map={'Poor':'red',
                              'Standard':'yellow',
                              'Good':'green'})
fig.update_traces(quartilemethod="exclusive")
fig.show()

data["Credit_Mix"] = data["Credit_Mix"].map({"Standard": 1,
                              "Good": 2,
                              "Bad": 0})
from sklearn.model_selection import train_test_split
x = np.array(data[["Annual_Income", "Monthly_Inhand_Salary",
                 "Num_Bank_Accounts", "Num_Credit_Card",
                 "Interest_Rate", "Num_of_Loan",
                 "Delay_from_due_date", "Num_of_Delayed_Payment",
                 "Credit_Mix", "Outstanding_Debt",
                 "Credit_History_Age", "Monthly_Balance"]])
#x=data.iloc[:,0:10]
x=np.array(x)

#y = np.array(data[["Credit_Score"]])
y=data. Credit_Score
xtrain, xtest, ytrain, ytest = train_test_split(x, y,
                                                test_size=0.33,
                                                random_state=42)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(xtrain, ytrain)
print("Credit Score Prediction : ")
a = float(input("Annual Income: "))
b = float(input("Monthly Inhand Salary: "))
c = float(input("Number of Bank Accounts: "))
d = float(input("Number of Credit cards: "))
```

```
e = float(input("Interest rate: "))
f = float(input("Number of Loans: "))
g = float(input("Average number of days delayed by the person: "))
h = float(input("Number of delayed payments: "))
i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")
j = float(input("Outstanding Debt: "))
k = float(input("Credit History Age: "))
l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])
print("Predicted Credit Score = ", model.predict(features))
```

Output:

```
Credit Score Prediction :
Annual Income: 100000
Monthly Inhand Salary: 50000
Number of Bank Accounts: 2
Number of Credit cards: 1
Interest rate: 7
Number of Loans: 2
Average number of days delayed by the person: 1
Number of delayed payments: 2
Credit Mix (Bad: 0, Standard: 1, Good: 3) : 1
Outstanding Debt: 1
Credit History Age: 2
Monthly Balance: 10000

Predicted Credit Score =  ['Standard']
```

Result:

Pgm.No.: 12 Iris Flower Classification using KNN
Date:

Aim:Iris Flower Classification using KNN using Machine Learning in Python

Procedure

KNN (K-Nearest Neighbors) is a classification algorithm that is widely used in machine learning. It is a simple and effective method for solving classification problems. The KNN algorithm works by classifying new data points based on the class of the K-nearest data points in the training set.
The Iris Flower Classification problem involves predicting the species of iris flowers based on their sepal length, sepal width, petal length, and petal width. The dataset contains 150 observations, each with 4 features and a corresponding species label (setosa, versicolor, or virginica).
To apply the KNN algorithm to the Iris Flower Classification problem, we first need to split the dataset into a training set and a test set. Then, we need to choose the value of K, which is the number of nearest neighbors to consider when making a classification decision. We can choose K by using a validation set or using cross-validation techniques.

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""
Iris Flower Classification using Python
"""

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
iris = pd.read_csv("D:/GEO/BE COURSES/2022 dec/LAB/DATASET/IRIS.csv")
#first five rows of this dataset:
print(iris.head())
print(iris.describe())
```

```
#The target labels of this dataset are present in the species column, let’s have a quick look at
the target labels:

print("Target Labels", iris["species"].unique())
#plot the data using a scatter plot which will plot the iris species according to the sepal length
and sepal width:

import plotly.io as io
io.renderers.default='browser'

import plotly.express as px
fig = px.scatter(iris, x="sepal_width", y="sepal_length", color="species")
fig.show()
#Iris Classification Model

x = iris.drop("species", axis=1)
y = iris["species"]
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.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train, y_train)

x_new = np.array([[5, 2.9, 1, 0.2]])
prediction = knn.predict(x_new)
print("Prediction: {}".format(prediction))
from sklearn import metrics
print("Prediction: {}".format(prediction))
```

Output:

```
Target Labels ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

Prediction: ['Iris-setosa']
```

Result:

Pgm.No.: 13 **Car Price Prediction**
Date:

AIM:Car Price Prediction Model using Python using KNN using Machine Learning in Python

Procedure:

Car Price Prediction is a regression problem that involves predicting the price of a car based on its features such as its make, model, year, mileage, and other specifications. The dataset contains a collection of observations, each with a set of features and a corresponding car price.
To apply the KNN algorithm to the Car Price Prediction problem, we first need to split the dataset into a training set and a test set. Then, we need to choose the value of K, which is the number of nearest neighbors to consider when making a regression prediction. We can choose K by using a validation set or using cross-validation techniques.
Once we have selected the value of K, we can use the training set to fit the KNN model. To predict the price of a new car, the KNN algorithm calculates the distance between the new car's features and all the cars in the training set. It then selects the K-nearest cars and predicts the price of the new car based on the average price of the K-nearest neighbors.

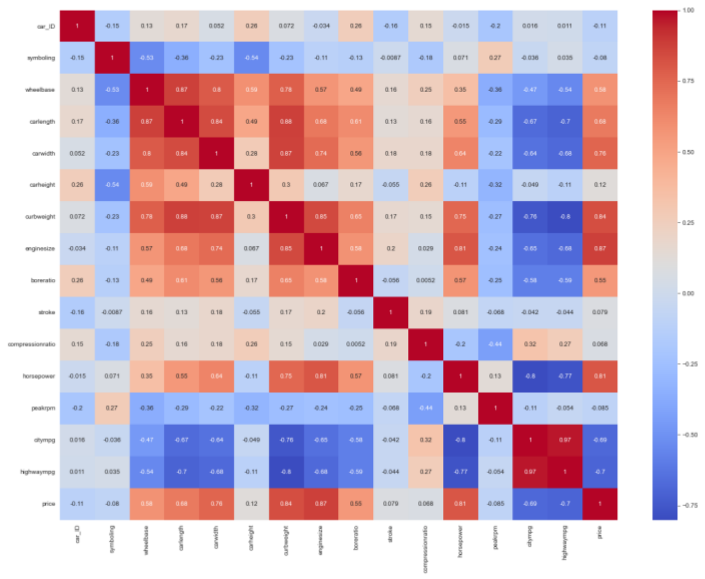
1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""
car price prediction
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

data = pd.read_csv("CarPrice.csv")
data.head()
data.shape
data.isnull().sum()
#So this dataset doesn’t have any null values, now let’s look at some of the other important insights to get
#an idea of what kind of data we’re dealing with:
data.info()
data.describe()
data.CarName.unique()

sns.set_style("whitegrid")
plt.figure(figsize=(15, 10))
sns.distplot(data.price)
plt.show()

#Now let’s have a look at the correlation among all the features of this dataset:

print(data.corr())
plt.figure(figsize=(20, 15))
correlations = data.corr()
sns.heatmap(correlations, cmap="coolwarm", annot=True)
plt.show()

#Training a Car Price Prediction Model
#predict = "price"

x=np.array(data[["symboling", "wheelbase", "carlength",
          "carwidth", "carheight", "curbweight",
          "enginesize", "boreratio", "stroke",
          "compressionratio", "horsepower", "peakrpm",
          "citympg", "highwaympg", "price"]])

y=np.array(data.price)

#x = np.array(data.drop([predict], 1))
#=data.iloc[:,0:15]
#y = np.array(data[predict])

from sklearn.model_selection import train_test_split
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)

from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(xtrain, ytrain)
predictions = model.predict(xtest)

from sklearn.metrics import mean_absolute_error
model.score(xtest, predictions)
```

Output:



Result:

Pgm.No.: 14 **House Price Prediction using KNN**
Date:

Aim:**House Price Prediction using Machine Learning in PythonProcedure**

Procedure:
House Price Prediction is a regression problem that involves predicting the price of a house based on its features such as its location, size, number of bedrooms, number of bathrooms, and other specifications. The dataset contains a collection of observations, each with a set of features and a corresponding house price.
To apply the KNN algorithm to the House Price Prediction problem, we first need to split the dataset into a training set and a test set. Then, we need to choose the value of K, which is the number of nearest neighbors to consider when making a regression prediction

Procedure:
1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

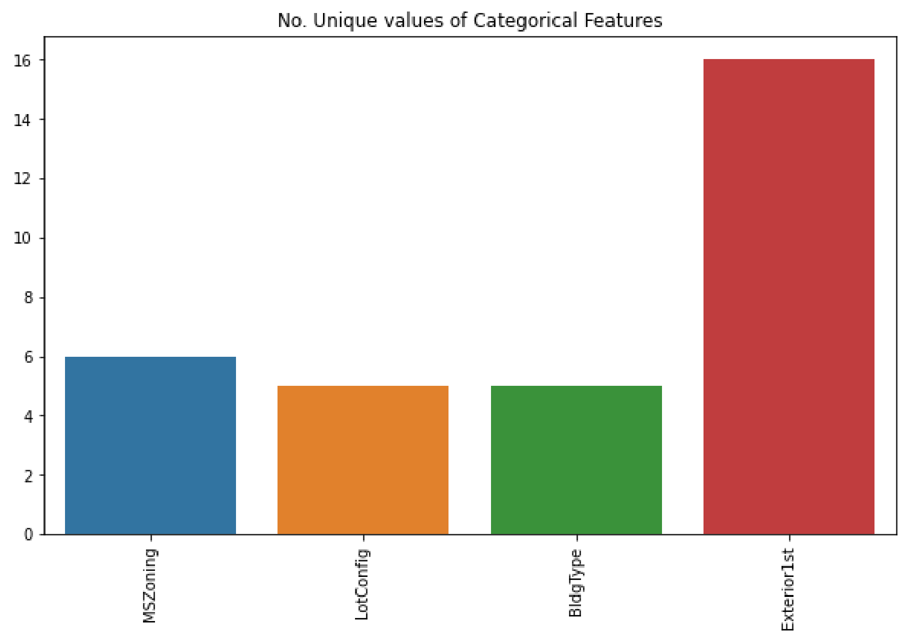
Program
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
dataset = pd.read_excel("HousePricePrediction.xlsx")

# Printing first 5 records of the dataset
print(dataset.head(5))
dataset.shape

obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
```

```
print("Float variables:",len(fl_cols))


plt.figure(figsize=(12, 6))
sns.heatmap(dataset.corr(),
                        cmap = 'BrBG',
                        fmt = '.2f',
                        linewidths = 2,
                        annot = True)

unique_values = []
for col in object_cols:
   unique_values.append(dataset[col].unique().size)
plt.figure(figsize=(10,6))
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)

plt.figure(figsize=(18, 36))
plt.title('Categorical Features: Distribution')
plt.xticks(rotation=90)
index = 1

for col in object_cols:
        y = dataset[col].value_counts()
        plt.subplot(11, 4, index)
        plt.xticks(rotation=90)
        sns.barplot(x=list(y.index), y=y)
        index += 1

dataset.drop(['Id'],
                        axis=1,
                        inplace=True)

dataset['SalePrice'] = dataset['SalePrice'].fillna(dataset['SalePrice'].mean())

new_dataset = dataset.dropna()

new_dataset.isnull().sum()


from sklearn.preprocessing import OneHotEncoder

s = (new_dataset.dtypes == 'object')
object_cols = list(s[s].index)
```

```
print("Categorical variables:")
print(object_cols)
print('No. of. categorical features: ',len(object_cols))

OH_encoder = OneHotEncoder(sparse=False)
OH_cols = pd.DataFrame(OH_encoder.fit_transform(new_dataset[object_cols]))
OH_cols.index = new_dataset.index
OH_cols.columns = OH_encoder.get_feature_names()
df_final = new_dataset.drop(object_cols, axis=1)
df_final = pd.concat([df_final, OH_cols], axis=1)


from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

X = df_final.drop(['SalePrice'], axis=1)
Y = df_final['SalePrice']

# Split the training set into
# training and validation set
X_train, X_valid, Y_train, Y_valid = train_test_split(X, Y, train_size=0.8, test_size=0.2,
random_state=0)
#Model and Accuracy
#svm

from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import mean_absolute_percentage_error

model_SVR = svm.SVR()
model_SVR.fit(X_train,Y_train)
Y_pred = model_SVR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))
#Random forest

from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor(n_estimators=10)
model_RFR.fit(X_train, Y_train)
Y_pred = model_RFR.predict(X_valid)

mean_absolute_percentage_error(Y_valid, Y_pred)

#LinearRegression
```

```
from sklearn.linear_model import LinearRegression

model_LR = LinearRegression()
model_LR.fit(X_train, Y_train)
Y_pred = model_LR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

Output:



```
0.1870512931870423
mean_absolute_percentage_error
0.18741683841599951
```

Result:

Pgm.No.: 15 Naïve Iris Classification
Date:

Aim:**Naïve Iris Classification using KNN using Machine Learning in Python**

Procedure:
The Naive Bayes algorithm is a classification algorithm based on Bayes' theorem, which describes the probability of an event occurring based on prior knowledge of conditions that might be related to the event. The Naive Bayes algorithm assumes that the features are conditionally independent, meaning that the presence of one feature does not affect the presence of another feature.

To apply the Naive Bayes algorithm to the Iris Flower Classification problem, we first need to split the dataset into a training set and a test set. Then, we can use the training set to fit the Naive Bayes model. The Naive Bayes algorithm calculates the probability of each feature for each class in the training set and uses Bayes' theorem to calculate the probability of each class given a new set of features.

Finally, we can evaluate the performance of the Naive Bayes model on the test set by calculating the accuracy, precision, recall, or F1-score, depending on the task. By tuning the hyperparameters of the Naive Bayes algorithm, such as the smoothing parameter, we can improve the performance of the algorithm on the Iris Flower Classification problem.

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program
```
from sklearn.naive_bayes import GaussianNB

from sklearn.naive_bayes import MultinomialNB

from sklearn import datasets

from sklearn.metrics import confusion_matrix

iris = datasets.load_iris()
```

```
gnb = GaussianNB()
mnb = MultinomialNB()

y_pred_gnb = gnb.fit(iris.data, iris.target).predict(iris.data)
cnf_matrix_gnb = confusion_matrix(iris.target, y_pred_gnb)
print('Gaussian Naive Bayes classifier Confusion Matrix')
print(cnf_matrix_gnb)

y_pred_mnb = mnb.fit(iris.data, iris.target).predict(iris.data)
cnf_matrix_mnb = confusion_matrix(iris.target, y_pred_mnb)
print('Multinomial Naive Bayes classifier Confusion Matrix')
print(cnf_matrix_mnb)
```

Output:

```
Gaussian Naive Bayes classifier Confusion Matrix
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
Multinomial Naive Bayes classifier Confusion Matrix
[[50  0  0]
 [ 0 46  4]
 [ 0  3 47]]
```

Result:

Pgm.No.: 16 **Comparison of Classification Algorithms**
Date:

Aim:Comparison of Classification Algorithms using KNN using Machine Learning in Python

Procedure:

When it comes to comparing different classification algorithms, there are several factors to consider such as accuracy, precision, recall, F1-score, training time, and computational complexity. Here is a brief comparison of the algorithms you mentioned:

Decision Tree Classifier: This algorithm builds a decision tree based on the features of the dataset, where each node represents a feature and each branch represents a decision based on that feature. Decision trees are easy to interpret, but they can easily overfit the training data. They are also sensitive to small variations in the data.

Logistic Regression: This algorithm models the probability of each class as a logistic function of the features. Logistic regression is a linear model that can handle non-linear relationships between the features and the target variable by using transformations such as polynomials or interactions. Logistic regression is easy to interpret, but it can suffer from multicollinearity and nonlinearity.

K-Nearest Neighbors Classifier: This algorithm predicts the class of a new sample based on the majority class of its k nearest neighbors in the training set. KNN is a non-parametric method that does not make assumptions about the distribution of the data. KNN can handle non-linear decision boundaries and can be easily adapted to multiclass problems. However, it can be computationally expensive, especially for large datasets, and it can be sensitive to the choice of k.Overall, the choice of classification algorithm depends on the specific problem and the characteristics of the dataset. It is often useful to try multiple algorithms and compare their performance using appropriate evaluation metrics.

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
import numpy
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.metrics import classification_report

iris= pd.read_csv("D:/GEO/BE COURSES/LAB/DATASET/IRIS.csv")
print(iris.head())

x = iris.drop("species", axis=1)
y = iris["species"]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,
y,test_size=0..10,random_state=42)

#x = np.array(data[["Age", "EstimatedSalary"]])
#y = np.array(data[["Purchased"]])

#xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.10,
random_state=42)
decisiontree = DecisionTreeClassifier()
logisticregression = LogisticRegression()
knearestclassifier = KNeighborsClassifier()
#svm_classifier = SVC()
bernoulli_naiveBayes = BernoulliNB()
passiveAggressive = PassiveAggressiveClassifier()

knearestclassifier.fit(x_train, y_train)
decisiontree.fit(x_train, y_train)
logisticregression.fit(x_train, y_train)
passiveAggressive.fit(x_train, y_train)
data1 = {"Classification Algorithms": ["KNN Classifier", "Decision Tree
Classifier",
                        "Logistic Regression", "Passive Aggressive Classifier"],
                "Score": [knearestclassifier.score(x,y), decisiontree.score(x, y),
                          logisticregression.score(x, y), passiveAggressive.score(x,y) ]}

score = pd.DataFrame(data1)
```

Output:

```
          Classification Algorithms     Score
          0                  KNN Classifier  0.953333
          1        Decision Tree Classifier  1.000000
          2             Logistic Regression  0.973333
          3  Passive Aggressive Classifier  0.800000
```

Result:

Pgm.No.: 17 Mobile Price Classification Algorithms
Date:

AIM:Mobile Price Classification Algorithms using KNN using Machine Learning in Python

Procedure:

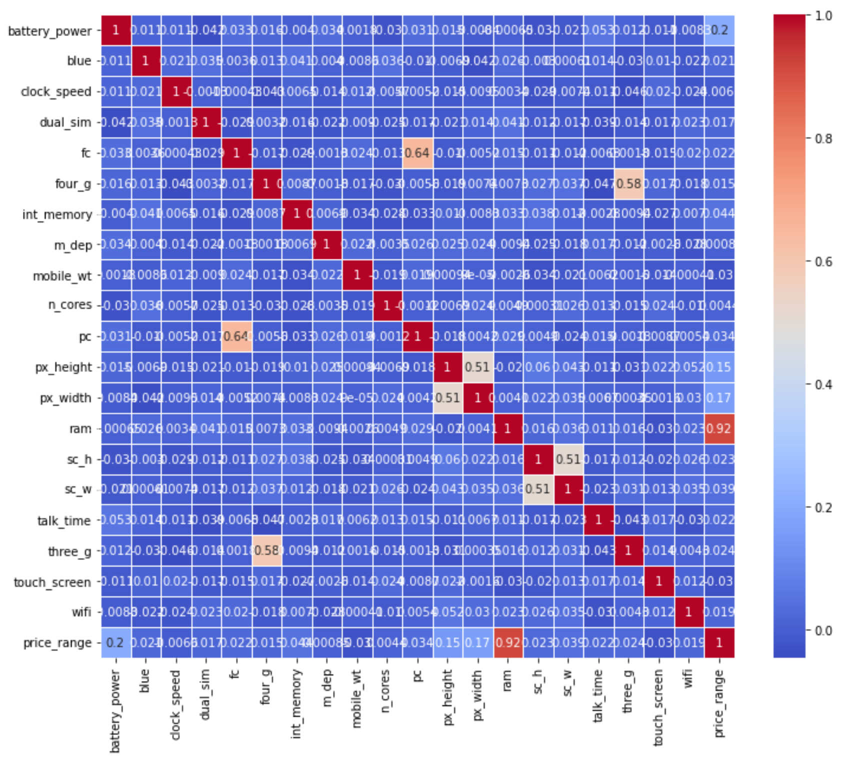
Mobile Price Classification using KNN is a common machine learning problem, where the goal is to predict the price range of a mobile phone based on its features such as RAM, battery capacity, camera resolution, etc. Here are the steps involved in solving this problem using KNN:

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""

Mobile Price Classification using Python

"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

#sns.set()


import plotly.io as io

io.renderers.default='browser'
```

```
data = pd.read_csv("mobile_prices.csv")
print(data.head())
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation
x = data.iloc[:, :-1].values
x=data.iloc[:,0:14]
#y = data.iloc[:, -1].values

y=data.price_range
#x = StandardScaler().fit_transform(x)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=0)

# Logistic Regression algorithm provided by Scikit-learn:
from sklearn.linear_model import LogisticRegression
lreg = LogisticRegression()
lreg.fit(x_train, y_train)
y_pred = lreg.predict(x_test)

 #accuracy of the model:
accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy of the Logistic Regression Model: ",accuracy)
#predictions made by the model:
print(y_pred)

 #Let's have a look at the number of mobile phones classified for each price range:
#(unique, counts) = np.unique(y_pred, return_counts=True)
price_range = np.asarray((unique, counts)).T
print(price_range)
```
Output:



```
Accuracy of the Logistic Regression Model:  64.25
[3 0 2 1 3 0 0 2 2 2 1 3 0 1 2 0 3 2 2 1 1 0 3 2 1 2 3 1 3 1 2 0 1 1 2 3 0
 0 3 2 3 3 3 3 2 2 0 1 3 1 0 2 0 3 0 3 3 1 0 3 3 2 2 1 1 3 3 3 2 2 3 2 1 0
 1 3 3 1 1 1 3 1 3 0 0 0 1 0 1 3 1 2 1 0 0 3 2 3 0 2 1 2 2 0 3 2 3 2 3 3 2
 0 0 2 3 3 1 0 1 0 0 3 2 2 1 2 0 0 0 3 1 3 3 2 3 3 3 3 0 1 1 3 2 3 0 3 0 0
 2 0 1 1 1 1 3 0 0 3 1 3 2 2 2 1 3 3 3 3 0 1 3 2 1 3 3 0 1 1 3 0 3 1 0 1 2
 1 3 0 3 3 3 3 1 1 2 1 0 2 2 0 3 3 3 0 1 3 2 2 0 0 0 2 2 2 0 0 0 2 2 2 2 3
 0 0 3 3 2 3 0 3 1 0 0 3 2 0 2 2 0 0 0 2 3 2 0 0 2 3 3 1 3 0 2 1 1 0 1 2 3
 2 0 0 1 3 3 3 2 3 3 2 1 2 2 2 1 3 2 2 2 1 0 2 1 0 0 0 1 3 3 3 0 1 2 0 1 2
 3 0 1 0 1 1 3 0 0 1 3 1 2 0 1 0 2 0 2 3 2 2 0 1 3 2 0 1 0 1 0 3 1 2 2 0 0
 2 3 0 3 1 2 0 1 3 0 2 1 1 2 2 2 0 2 0 0 2 1 2 3 2 2 0 3 2 3 2 2 2 2 3 3 0
 2 1 1 0 1 1 2 2 0 3 1 1 0 1 1 3 1 0 0 3 0 3 0 2 3 1 1 0 2 1]
```

Result:

Pgm.No.: 18 **Perceptron IRIS Classification Algorithms**
Date:

Aim:Perceptron IRIS Classification Algorithms using KNN using Machine Learning in Python

Procedure:

Perceptron is a simple and popular algorithm used for binary classification problems. It is also used as a building block for more complex algorithms such as neural networks. Here are the steps involved in solving the IRIS classification problem using the perceptron algorithm:

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""
Perceptron IRIS classiification
"""
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
```

```
X = iris.data[:, [2, 3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)

y_pred = ppn.predict(X_test_std)

print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Accuracy: %.3f' % ppn.score(X_test_std, y_test))
```

Output:

```
perceptron IRIS classification Accuracy: 0.978
```

Result:

Pgm.No.: 19 **Implementation of Naive Bayes Algorithms**
Date:

Aim:Implementation of Naive Bayes in Python – Machine Learning

Procedure:

Naive Bayes in Python for breast cancer classification. The necessary libraries such as numpy and pandas are imported, and the breast cancer dataset is read using pandas. The dataset is then split into training and testing sets using the train_test_split function from sklearn. Feature scaling is performed on the dataset using the StandardScaler method from sklearn.preprocessing.
The Naive Bayes Classification model is trained on the training set using the GaussianNB function from sklearn.naive_bayes library. The hyperparameters such as kernel and random_state are set to linear and 0 respectively, and the remaining hyperparameters are set to their default values.
Finally, the model is evaluated using evaluation metrics such as confusion matrix and accuracy, which are imported from sklearn.metrics. The predictions are made on the testing set, and the confusion matrix and accuracy are printed.

Procedure:

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""

Implementation of Naive Bayes in Python – Machine Learning

In this tutorial, we will understand the Implementation of Naive Bayes in Python – Machine

Learning.

Importing the Necessary libraries

"""

import numpy as np

import pandas as pd
```

```
#Importing the dataset
#read the dataset.
dataset = pd.read_csv("breastcancer.csv")
#X = dataset.iloc[:, :-1].values
#y = dataset.iloc[:, -1].values
X=dataset.iloc[:,0:10]
y=dataset.Class
#splitting dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
#Feature Scaling
"""
Feature scaling is the process of converting the data into a min-max range. In this case,
 the standard scalar method is used.
"""
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
#Training the Naive Bayes Classification model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
#Naive Bayes classifier model
GaussianNB(priors=None, var_smoothing=1e-09)
#Display the results (confusion matrix and accuracy)
"""
Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the
performance of the model built using a decision tree classifier.
"""
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Output:

```
Confusion matrix of Classifier
[[99  8]
 [ 2 62]]
```

Result:

Pgm.No.: 20 **Future Sales Prediction**

Date:

Aim:Future Sales Prediction using KNN using Machine Learning in Python

Procedure:

Future sales prediction using Linear Regression in Python. It imports necessary libraries such as pandas, numpy, and LinearRegression from the sklearn.linear_model library. It reads the dataset "futuresale prediction.csv" using the pd.read_csv() function and prints the first five rows of the dataset using the head() function.It also prints the five randomly sampled rows of the dataset using the sample() function and checks for any null values using the isnull().sum() function.The code then creates three scatter plots using the plotly.express library to visualize the correlation between sales and TV, Newspaper, and Radio advertisements.

The code then calculates the correlation between sales and the other variables using the corr() function and prints the correlation values sorted in descending order.It then splits the data into training and testing sets using the train_test_split() function from sklearn.model_selection.Next, the code creates an instance of LinearRegression and trains the model using the fit() function. It then calculates the accuracy of the model using the score() function.Finally, the code predicts the future sales using the predict() function for given values of TV, Radio, and Newspap

1. Import all libraries
2. Read the dataset
3. Data exploration
4. Split the dataset for testing and training
5. Load the model for training
6. Train the model
7. Find out the result

Program

```
"""

Future Sales Prediction using Python

"""


import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import plotly.io as io
io.renderers.default='browser'

data = pd.read_csv("futuresale prediction.csv")
print(data.head())
print(data.sample(5))
print(data.isnull().sum())

import plotly.express as px
import plotly.graph_objects as go
figure = px.scatter(data_frame = data, x="Sales",
                    y="TV", size="TV", trendline="ols")
figure.show()
figure = px.scatter(data_frame = data, x="Sales",
                    y="Newspaper", size="Newspaper", trendline="ols")
figure.show()
figure = px.scatter(data_frame = data, x="Sales",
                    y="Radio", size="Radio", trendline="ols")
figure.show()

correlation = data.corr()
print(correlation["Sales"].sort_values(ascending=False))
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

x = np.array(data.drop(labels=["Sales"], axis=1))  # Specify the 'axis' argument
```

```
y = np.array(data["Sales"])  # Specify the column using 'labels'

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=42)
model = LinearRegression()
model.fit(xtrain, ytrain)
print('Future sale Model Accuracy')
print(model.score(xtest, ytest))
#features = [[TV, Radio, Newspaper]]
features = np.array([[230.1, 37.8, 69.2]])
print(model.predict(features))
```

Output:

```
Future sale Model Accuracy
0.9059011844150826
Future sale prediction
[21.37254028]
```

Result: