Practical No: 1

Design the Machine Learning Model

AIM: Design a simple machine learning model to train the training instances and test the same.

Description:

1. Training Data

Training data is the data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict.

Training data is always more or equal in size than test data

2. Test Data

Testing data is used to evaluate our model performance.

Code with output

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
x = numpy.random.normal(3,1,100)
print(x)
y = numpy.random.normal(150,40,100) /x
print(y)
plt.scatter(x,y)
plt.show()
```

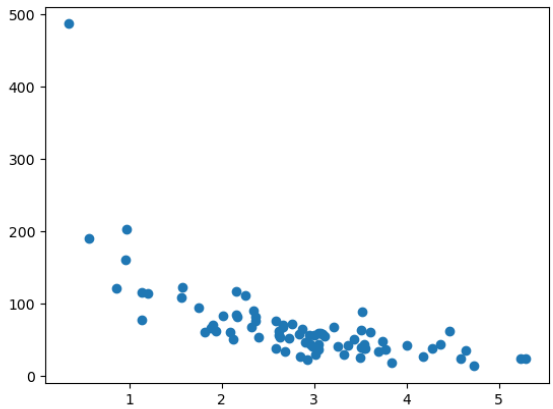
```
In [2]: runcell(0, 'D:/Python/NK12.py')
[2.58324215 2.94373317 0.8638039  4.64027081 1.20656441 2.15825263
 3.50288142 1.75471191 1.94204778 2.09099239 3.55145404 5.29220801
 3.04153939 1.88207455 3.53905832 2.4038403  2.9808695  4.17500122
 2.25212905 3.00902525 2.12189211 2.84356583 3.25657045 2.01122095
 2.66117803 2.76381597 2.36234499 1.81238771 1.57878277 2.8465048
 2.73094304 5.23136679 0.56523242 3.1127265  3.37044454 4.35963386
 3.50185721 2.1557863  3.00000976 3.54235257 2.6864918  3.77101174
 1.13190935 4.73118467 4.46767801 2.66432266 3.61134078 3.04797059
 2.17086471 3.08771022 4.00036589 2.61890748 2.62433058 2.92552924
 3.43349633 4.27837923 2.36532069 3.50839624 3.21611601 1.14138761
 2.58068352 2.8676711  2.96042976 3.32600343 0.95967695 3.04625552
 2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
 2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
 3.69511961 0.96653345 2.81053074 2.92278133 3.82470301 4.24821292
 2.59610773 1.61548133 4.36723542 4.21788563 2.53799465 3.35088849
 3.38186623 3.56627544 3.20420798 4.40669624 1.2620405  4.04082395
 3.38047197 2.78286473 4.1735315  0.65639681]
[ 76.05204933  56.20180641 121.17874037  36.05903817 114.23885932
 117.41526024  63.77986643  95.52998052  62.4237197   60.57574247
  38.57519009  24.10914678  37.45148182  67.13926856  39.26265343
  53.79918302  40.94657678  27.02857247 111.90190427  30.26663537
  51.4368334   58.83311239  42.08623741  83.01076429  68.37843898
  72.54627253  76.22874513  60.83111238 123.11113005  27.89501382
  53.25015791  24.86406278 190.30762228  55.79245737  42.32964984
  43.76381026  25.90093643  85.28325651  56.63901768  43.77321677
  34.70979433  37.10649687  77.86225629  14.09666443  62.93869329
  70.87521926  61.39097018  43.58292288  81.92492065  57.61442568
  43.5111781   57.3316853   53.67848811  22.97550427  50.79538368
  39.01941998  82.32095959  39.62788318  68.30365792 115.73628743
  38.66530343  65.39332448  44.34023444  30.00934597 161.50533328
  59.1743156   68.74904453 108.8692008   89.19445659  48.95077634
  90.02681869  18.36485932  62.86162946  59.01318439  71.22685026
  25.07604874 487.03726791  47.24533754  34.16662793 202.76589695
  72.37873053  55.46264153  34.46826737  40.15213735  70.55883508
 108.46604975  21.035144    32.35727584  64.76189111  52.19177448
  55.71813453  50.5667094   32.65308038  27.61777936  80.14230427
  54.98360439  46.50723143  61.85229524  45.84155234 208.47130994]
```

train_x = x[:80]
train_y = y[:80]

test_x = x[:20]
test_y = y[:20]

print(train_x,train_y,test_x,test_y)

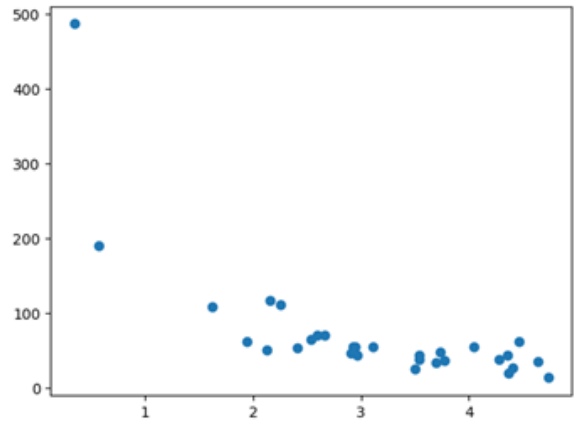
```
QT_SCALE_FACTOR to set the application global scale factor.
[2.58324215 2.94373317 0.8638039  4.64027081 1.20656441 2.15825263
 3.50288142 1.75471191 1.94204778 2.09099239 3.55145404 5.29220801
 3.04153939 1.88207455 3.53905832 2.4038403  2.9808695  4.17500122
 2.25212905 3.00902525 2.12189211 2.84356583 3.25657045 2.01122095
 2.66117803 2.76381597 2.36234499 1.81238771 1.57878277 2.8465048
 2.73094304 5.23136679 0.56523242 3.1127265  3.37044454 4.35963386
 3.50185721 2.1557863  3.00000976 3.54235257 2.6864918  3.77101174
 1.13190935 4.73118467 4.46767801 2.66432266 3.61134078 3.04797059
 2.17086471 3.08771022 4.00036589 2.61890748 2.62433058 2.92552924
 3.43349633 4.27837923 2.36532069 3.50839624 3.21611601 1.14138761
 2.58068352 2.8676711  2.96042976 3.32600343 0.95967695 3.04625552
 2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
 2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
 3.69511961 0.96653345] [ 76.05204933  56.20180641 121.17874037  36.05903817 114.23885932
```



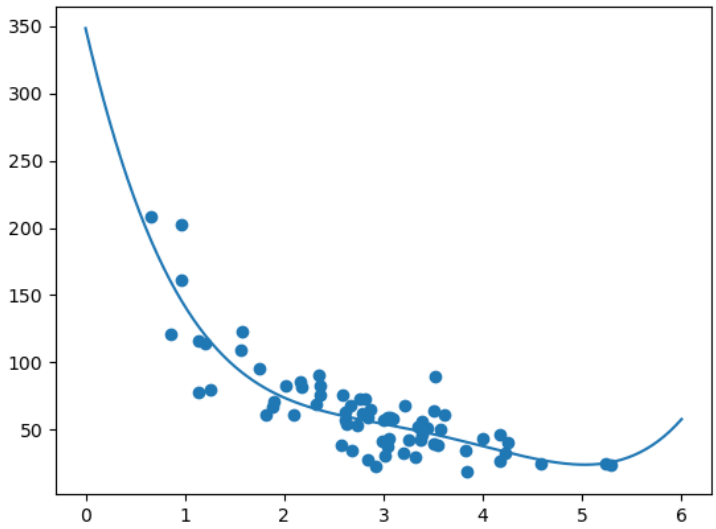
plt.scatter(train_x,train_y)
plt.show()

train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.3)

plt.scatter(test_x,test_y)
plt.show()

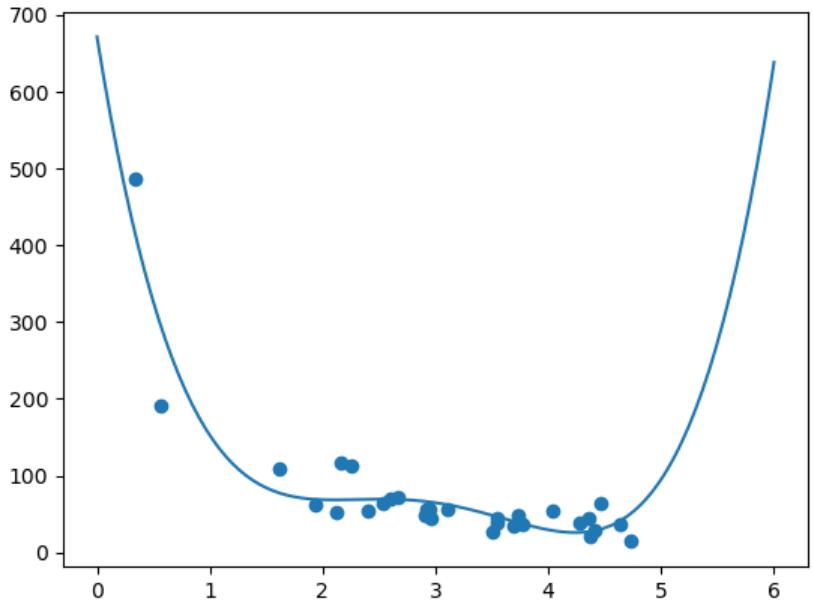


```
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0,6,200)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```




```
mymodel = numpy.poly1d(numpy.polyfit(test_x, test_y, 4))
myline = numpy.linspace(0,6,200)
plt.scatter(test_x, test_y)
plt.plot(myline, mymodel(myline))
plt.show()

r2 = r2_score(train_y, mymodel(train_x))
print(r2)
print(mymodel(5))
```




```
0.19835294359936562
95.12966899800244
```

As we get high rscore the model is working good

Learnings

First we have created random data for x and y.

Then we have divided it into train test part with 80:20 ratio. visualizes the data and the fitted models.

Then after fitting model, we have evaluated model performance using r square. Then make prediction using trained model

Practical No: 2

Concept Learning

AIM: Implement and demonstrate the find-s algorithm for finding the most specific.

Description:

1. Training dataset table (input data):

	A	B	C	D	E	F	G
1	sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							

2.: Write the right hypothesis/function from historical data
One of the often-used statistical concepts in machine learning is the hypothesis. It is notably employed in supervised machine learning, where an ML model uses a dataset to train a function that most effectively translates input to related outputs.

In this code person enjoys sport if weather is sunny, airtemp is warm, wind is strong

3. How Does It Work?
It eliminates attribute that do not affect target column

Code with output

```
import csv
num_attributes = 6
a = []

print("\n The Given Training Dataset \n")
with open('Book1.csv','r') as csvfile:
  reader = csv.reader(csvfile)
  count = 0
  for row in reader:
    if count == 0:
      print(row)
      count+=1;
    else:
      a.append(row)
      print(row)
```

```
    count+=1

print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

for j in range(0,num_attributes):
  hypothesis[j]= a[0][j];
  print(hypothesis)

print("\n find S:finding a Maximally specific Hypothesis\n")
for i in range(0,len(a)):
  if a[i][num_attributes]=="Yes":
    for j in range(0,num_attributes):
      if a[i][j]!=hypothesis[j]:
        hypothesis[j]='?'
      else:
        hypothesis[j] = a[i][j]
  print("for training example no :{0} the hypothesis is".format(i),hypothesis)
```

```
The Given Training Dataset

['sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']
['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 initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
```

```
['Sunny', '0', '0', '0', '0', '0']
['Sunny', 'Warm', '0', '0', '0', '0']
['Sunny', 'Warm', 'Normal', '0', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

```
find S:finding a Maximally specific Hypothesis

for training example no :0 the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
for training example no :1 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
for training example no :2 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
for training example no :3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
import csv
a = []
with open('book2.csv', 'r') as csvfile:
    next(csvfile)
    for row in csv.reader(csvfile):
        a.append(row)

for x in a:
  print(x)

print("\nThe total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\nThe initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        print ("\nInstance ", i+1, "is", a[i], " and is Positive Instance")
        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("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
    if a[i][num_attribute] == 'no':
        print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")
        print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)
```

```
['some', 'small', 'no', 'affordable', 'many', 'no']
['many', 'big', 'no', 'expensive', 'one', 'yes']
['some', 'big', 'always', 'expensive', 'few', 'no']
['many', 'medium', 'no', 'expensive', 'many', 'yes']
['many', 'small', 'no', 'affordable', 'many', 'yes']
```

```
The total number of training instances are :  5
```

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

```
Instance  1 is ['some', 'small', 'no', 'affordable', 'many', 'no']  and is Negative Instance Hence Ignored
The hypothesis for the training instance 1  is:  ['many', '?', 'no', '?', '?']


Instance  2 is ['many', 'big', 'no', 'expensive', 'one', 'yes']  and is Positive Instance
The hypothesis for the training instance 2  is:  ['many', '?', 'no', '?', '?']


Instance  3 is ['some', 'big', 'always', 'expensive', 'few', 'no']  and is Negative Instance Hence Ignored
The hypothesis for the training instance 3  is:  ['many', '?', 'no', '?', '?']


Instance  4 is ['many', 'medium', 'no', 'expensive', 'many', 'yes']  and is Positive Instance
The hypothesis for the training instance 4  is:  ['many', '?', 'no', '?', '?']


Instance  5 is ['many', 'small', 'no', 'affordable', 'many', 'yes']  and is Positive Instance
The hypothesis for the training instance 5  is:  ['many', '?', 'no', '?', '?']
```

```
The Maximally specific hypothesis for the training instance is  ['many', '?', 'no', '?', '?']
```

Learnings

This Python code reads data from a CSV file and uses the Find-S algorithm for binary classification. It iterates through training instances, adjusting a hypothesis to correctly classify positive cases while minimizing errors. When negative cases are encountered, conflicting attributes are marked with '?' to ensure accuracy. The resulting 'hypothesis' is the most specific rule for the given training data.

Practical No: 3

Multiclass classification (Problem based Learning)

AIM: Support vector machine (SVM) algorithm for multiclass classification using Iris.csv and wine dataset from sklearn.

Description:

Calculate the TP, TN, FP, FN values for the class Setosa using the confusion matrix / contingency table and also calculate precision and recall for data file 'wine' from sklearn dataset:

TP (True Positives): The number of correctly predicted positive instances in a binary classification problem.

TN (True Negatives): The number of correctly predicted negative instances in a binary classification problem.

FP (False Positives): The number of instances that were predicted as positive but are actually negative in a binary classification problem.

FN (False Negatives): The number of instances that were predicted as negative but are actually positive in a binary classification problem.

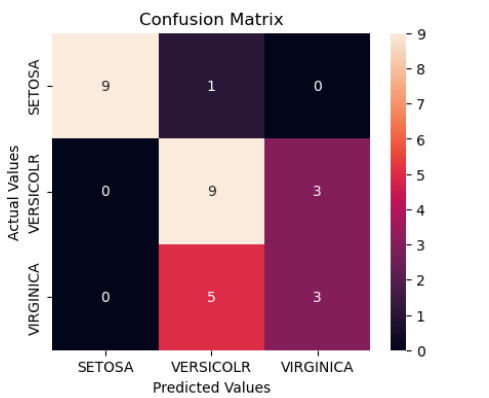
Support Vector Machine (SVM): A supervised machine learning algorithm that finds a hyperplane to maximize the margin between different classes in a dataset, making it effective for classification and regression tasks.

Code with output

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
iris = datasets.load_iris()
#print(iris.data)
X = iris.data[:, :2]
```

```
y = iris.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, train_size=0.80,
test_size=0.20, random_state=101)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
poly_accuracy = accuracy_score(y_test, poly_pred)
poly_f1 = f1_score(y_test, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
rbf_accuracy = accuracy_score(y_test, rbf_pred)
rbf_f1 = f1_score(y_test, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
cm = confusion_matrix(y_test,poly_pred)
cm_df = pd.DataFrame(cm,
                     index = ['SETOSA','VERSICOLR','VIRGINICA'],
                     columns = ['SETOSA','VERSICOLR','VIRGINICA'])
plt.figure(figsize=(5,4))
#print(cm_df)
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
Accuracy (RBF Kernel):  76.67
F1 (RBF Kernel):  76.36
```



Code and output for wine dataset from sklearn:

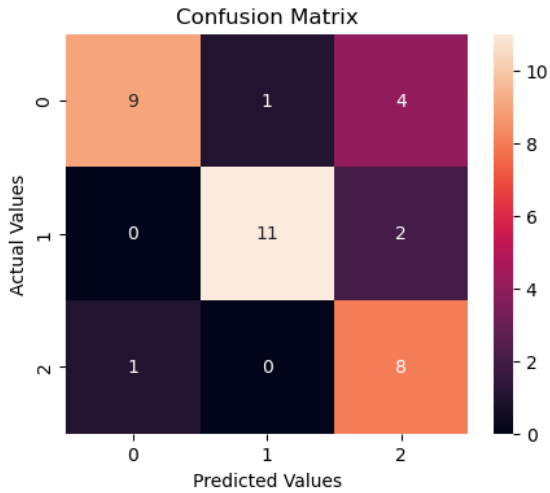
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
wine = datasets.load_wine()
X = wine.data[:, :2]
y = wine.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, train_size=0.80,
test_size=0.20, random_state=101)
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
poly_accuracy = accuracy_score(y_test, poly_pred)
poly_f1 = f1_score(y_test, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
rbf_accuracy = accuracy_score(y_test, rbf_pred)
rbf_f1 = f1_score(y_test, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
cm = confusion_matrix(y_test,poly_pred)
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(5,4))
#print(cm_df)
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

OUTPUT

```
Accuracy (Polynomial Kernel):  77.78
F1 (Polynomial Kernel):  78.34
```

```
Accuracy (RBF Kernel):  77.78
F1 (RBF Kernel):  79.18
```

```
plt.show()
```



Learnings

The provided code performs machine learning classification using two SVM kernels (Polynomial and RBF) on the wine dataset.

The code also generates and visualizes a confusion matrix, which is a valuable tool for evaluating the classification model's performance. The confusion matrix displays the number of true positive (correctly predicted positive class), true negative (correctly predicted negative class), false positive (predicted positive but actual negative), and false negative (predicted negative but actual positive) instances. It provides a clear summary of the model's classification accuracy and potential errors, helping to assess its strengths and weaknesses in differentiating between the wine categories.

Practical No: 4

Candidate-elimination algorithm

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.

Description:

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified from generalize form.

Terms :-

General Hypothesis: Not Specifying features to learn the machine.

G = {'?', '?','?','?'…}: Number of attributes.

Specific Hypothesis: Specifying features to learn machine (Specific feature).

S= {'pi','pi','pi'…}: Number of pi depends on number of attributes.

Version Space: It is intermediate of general hypothesis and Specific hypothesis. It not only just written one hypothesis but a set of all possible hypothesis based on training data-set.

Candidate-elimination algorithm :-

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

```
if attribute_value == hypothesis_value:
```

Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

Make generalize hypothesis more specific.

Code and output :-

```
import numpy as np
import pandas as pd

#Loading data from a csv file.
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
print(data)
```

```
[1]  ✓ 4.4s

...        sky air_temp humidity    wind water forecast enjoy_sport
     0  sunny     warm   normal  strong  warm     same         yes
     1  sunny     warm     high  strong  warm     same         yes
     2  rainy     cold     high  strong  warm   change          no
     3  sunny     warm     high  strong  cool   change         yes
```

```
#Separating concept features from Target
concepts = np.array(data.iloc[:,0:6])
print(concepts)
```

```
[2]  ✓ 0.0s

...  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
      ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
      ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
      ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

```
#Isolating target into a separate DataFrame
#Copying last column to target  array
target = np.array(data.iloc[:,6])
print(target)
```

```
[3]  ✓ 0.0s

...  ['yes' 'yes' 'no' 'yes']
```

```
def learn(concepts, target):
#Initialise S0 with the first instance from concepts.
#.copy()makes sure a new list is created instead of just pointing to the same memory location.
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)
# The learning iterations.
    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
# Checking if the hypothesis has a positive target.
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
# Change values in S & G only if values change.
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
# Checking if the hypothesis has a positive target.
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
# For negative hypothesis change values only in G.
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")
# find indices where we have empty rows, meaning those that are unchanged.
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
# remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])
# Return final values
    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")
```

```
...
Initialization of specific_h and genearal_h

Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'


Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
```

```
Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?',


Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?',


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

Practical No: 5

Naive Bayes and Gaussian Classification

AIM: Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Description:

Naïve Bayesian classifier:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Gaussian Classifier :

A Gaussian classifier, often known as a Gaussian Naive Bayes classifier, is a method of classification that uses this distribution to predict results by assuming that the features have a Gaussian (normal) distribution. This approach is frequently employed in situations requiring continuous numerical data.

Code with output

```
import numpy as np
import pandas as pd
import sklearn
#Import dataset
from sklearn import datasets
wine = datasets.load_wine()
print("Features: ", wine.feature_names)
print("Labels: ", wine.target_names)

X=pd.DataFrame(wine['data'])
print(X.head())
print(wine.data.shape)
y=print(wine.target)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target,
test_size=0.30,random_state=10)
#import gaussian naive bayes model.
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train,y_train)

#predict the response for test dataset
y_pred = gnb.predict(X_test)
print(y_pred)

from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm
```

OUTPUT

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
Labels:  ['class_0' 'class_1' 'class_2']
      0     1     2     3      4    5     6     7     8     9     10    11  \
0  14.23  1.71  2.43  15.6  127.0  2.80  3.06  0.28  2.29  5.64  1.04  3.92
1  13.20  1.78  2.14  11.2  100.0  2.65  2.76  0.26  1.28  4.38  1.05  3.40
2  13.16  2.36  2.67  18.6  101.0  2.80  3.24  0.30  2.81  5.68  1.03  3.17
3  14.37  1.95  2.50  16.8  113.0  3.85  3.49  0.24  2.18  7.80  0.86  3.45
4  13.24  2.59  2.87  21.0  118.0  2.80  2.69  0.39  1.82  4.32  1.04  2.93

      12
0  1065.0
1  1050.0
2  1185.0
3  1480.0
4   735.0
(178, 13)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```
[3]  GaussianNB
     GaussianNB()
```

```
[4]  #predict the response for test dataset
     y_pred = gnb.predict(X_test)
     print(y_pred)

[1 1 0 1 0 1 1 0 2 0 0 0 0 2 1 1 2 1 1 2 0 2 0 0 1 2 1 2 1 1 2 2 2 2 1 0 0
 1 0 1 1 0 2 1 0 1 2 1 1 1 2 1 2 0]
```

```
5    cm
```

```
Accuracy: 0.888888888888888
array([[14,  1,  0],
       [ 2, 22,  3],
       [ 0,  0, 12]])
```

Analysis of Confusion Matrix

- Row 1 (True Class 1):

  14 instances of Class 1 were correctly predicted as Class 1 (True Positives).

  1 instance of Class 1 was incorrectly predicted as Class 2 (False Negative).

  0 instances of Class 1 were incorrectly predicted as Class 3 (False Negative).

- Row 2 (True Class 2):

  2 instances of Class 2 were incorrectly predicted as Class 1 (False Positive).

22 instances of Class 2 were correctly predicted as Class 2 (True Positives).

3 instances of Class 2 were incorrectly predicted as Class 3 (False Negative).

- Row 3 (True Class 3):

  0 instances of Class 3 were incorrectly predicted as Class 1 (False Positive).

  0 instances of Class 3 were incorrectly predicted as Class 2 (False Positive).

  12 instances of Class 3 were correctly predicted as Class 3 (True Positives).

Learnings

1. It loads the "wine" dataset, which is a standard dataset available in scikit-learn containing information about different types of wines.
2. It splits the dataset into training and testing sets.
3. It trains a Gaussian Naive Bayes classifier on the training data.
4. The classifier is used to make predictions on the test data.
5. The code calculates and prints the accuracy of the classifier's predictions.
6. It also computes and displays the confusion matrix, which provides information about how well the classifier performed in terms of correctly classifying different wine types.

Practical No: 6

Decision Tree Classifier & Random Forest Classifier

AIM: Write a program to implement the Decision Tree Classifier & Random Forest Classifier with prediction, test score and confusion matrix.

Description:

Decision Tree Classifier:

Interpretability: Decision trees offer easy interpretability, aiding in understanding and explaining the logic behind classification decisions.

Overfitting: Decision trees can be prone to overfitting, especially if deep or complex, necessitating regularization techniques for optimal performance.

Random Forest Classifier:

Ensemble Learning: Random Forest is an ensemble method that combines multiple decision trees, enhancing model accuracy and stability.

Variance Reduction: Random Forest reduces variance by aggregating predictions from different trees, mitigating overfitting and improving generalization to new data.

Code with output

```
import pandas as pd
 import numpy as np
 import matplotlib.pyplot as plt
 import seaborn as sns
 from sklearn.preprocessing import LabelEncoder
 from sklearn.model_selection import train_test_split
 from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
 %matplotlib inline

df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")

# Keeping emp position unaffected.
 df.head()
```

```
# Exploratory Data Analysis
 sns.countplot(x='Attrition', data=df)

from pandas.core.arrays import categorical

df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis="columns",
inplace=True)

categorical_col = []

for column in df.columns:
    if df[column].dtype == object:
        categorical_col.append(column)

df['Attrition'] = df['Attrition'].astype("category").cat.codes

for column in categorical_col:
    df[column] = LabelEncoder().fit_transform(df[column])

X = df.drop('Attrition', axis=1)
 y = df['Attrition']

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

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n=====================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print(" ")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print(" ")
        print(f"Confusion Matrix: \n{confusion_matrix(y_train, pred)}\n")
```

```
    elif not train:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True)
        )
        print("Test Result:\n========================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print(" ")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print(" ")
        print(f"Confusion Matrix: \n{confusion_matrix(y_test, pred)}\n")
from sklearn.tree import DecisionTreeClassifier
 from sklearn.ensemble import RandomForestClassifier
 from pickle import TRUE

from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(random_state=42)
 tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
 print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)

from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42)
 rf_clf.fit(X_train, y_train)

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
 print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

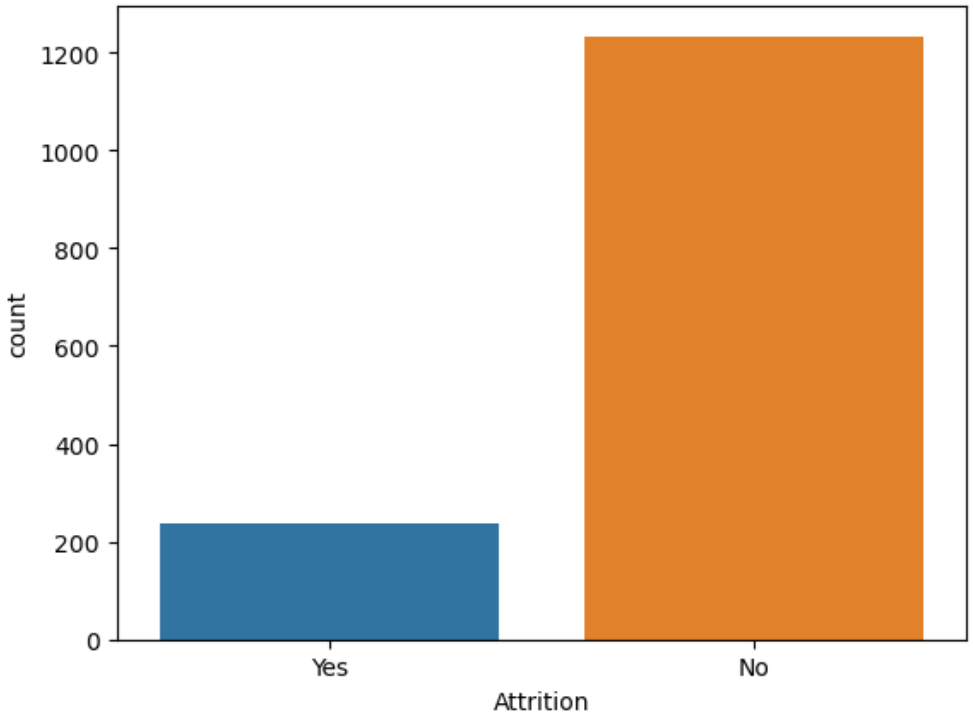
OUTPUT

In [3]:
```
df.head()
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	Rela
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	

5 rows × 35 columns



```
Train Result:
========================================
Accuracy Score: 100.00%

CLASSIFICATION REPORT:
              0      1  accuracy  macro avg  weighted avg
precision   1.0    1.0       1.0        1.0           1.0
recall      1.0    1.0       1.0        1.0           1.0
f1-score    1.0    1.0       1.0        1.0           1.0
support   853.0  176.0       1.0     1029.0        1029.0

Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
========================================
Accuracy Score: 77.78%

CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.887363   0.259740  0.777778    0.573551      0.800549
recall      0.850000   0.327869  0.777778    0.588934      0.777778
f1-score    0.868280   0.289855  0.777778    0.579067      0.788271
support   380.000000  61.000000  0.777778  441.000000    441.000000

Confusion Matrix:
[[323  57]
 [ 41  20]]
```

```
Train Result:
========================================
Accuracy Score: 100.00%

CLASSIFICATION REPORT:
              0      1  accuracy  macro avg  weighted avg
precision   1.0    1.0       1.0        1.0           1.0
recall      1.0    1.0       1.0        1.0           1.0
f1-score    1.0    1.0       1.0        1.0           1.0
support   853.0  176.0       1.0     1029.0        1029.0

Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
========================================
Accuracy Score: 86.17%

CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.871795   0.500000  0.861678    0.685897      0.820367
recall      0.984211   0.098361  0.861678    0.541286      0.861678
f1-score    0.924598   0.164384  0.861678    0.544491      0.819444
support   380.000000  61.000000  0.861678  441.000000    441.000000

Confusion Matrix:
[[374   6]
 [ 55   6]]
```

Confusion Matrix Calculations:

Ninad Karlekar 22306A1012

Random Forest confusion matrix

	Attrition	No Attrition
Predicted +	374 (TP)	6 (FP)
predicted -	55 (FN)	6 (TN)

1 Accuracy $= \frac{TP + TN}{TP + FP + FN + TN} = \frac{380}{441}$

$= 86.16\%$

2 Precision $= \frac{TP}{TP+FP} = \frac{374}{374+6} = 98\%$

3 Recall $= \frac{TP}{TP+FN} = \frac{374}{374+55} = 87\%$

Decision tree confusion matrix

$$\begin{bmatrix} 323 \text{ (TP)} & 57 \text{ (FP)} \\ 41 \text{ (FN)} & 20 \text{ (TN)} \end{bmatrix}$$

Acc $= \frac{323+20}{323+57+41+20} = \frac{343}{441} = 77\%$

Precision $= \frac{323}{323+57} = \frac{323}{380} = 85\%$

Recall $= \frac{323}{323+41} = \frac{323}{364} = 88\%$

Analysis of Confusion Matrix

The model correctly identified 6 instances as positive.

It correctly identified 374 instances as negative.

However, it made 6 false positive predictions, indicating instances that were predicted as positive but were actually negative.

It also made 55 false negative predictions, indicating instances that were predicted as negative but were actually positive.

Practical No: 7

Feature Selection

AIM: Data loading, feature scoring and ranking, feature selection (principal component analysis)

Description:

Principal Component Analysis:

Principal Component Analysis (PCA) is a handy tool in statistics for making complex data easier to understand. Before diving into it, it's crucial to standardize the data, which basically means making sure all the different measurements are on the same scale. Next, PCA calculates something called the covariance matrix, which shows how different variables in the data relate to each other. Then comes the interesting part: eigendecomposition. This involves finding special directions, called eigenvectors, and their associated importance values, called eigenvalues. The eigenvectors with the highest eigenvalues become the principal components, kind of like the main characters in our data story. These principal components capture the most important aspects of the data. By selecting a few of these components, we can simplify the data without losing much information. People often use PCA to make big datasets more manageable, create cool visualizations, and reduce noise in the data. It's like turning a complicated puzzle into a simpler picture, making it easier for us to see the patterns and trends.

Code with output

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(url, names=names)
dataset.head()

# Store the feature sets into X variable and the series of corresponding variables in y
x = dataset.drop('Class', axis=1)
y = dataset['Class']
x.head()
y.head()

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

sc = StandardScaler()
x_train1 = sc.fit_transform(x_train)
x_test1 = sc.transform(x_test)

y_train1 = y_train
y_test1 = y_test

pca = PCA()
x_train1 = pca.fit_transform(x_train1)
x_test1 = pca.transform(x_test1)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)

pca = PCA(n_components=1)
x_train1 = pca.fit_transform(x_train1)
x_test1 = pca.transform(x_test1)

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(x_train1, y_train1)
```

```
y_pred = classifier.predict(x_test1)

cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy:', accuracy_score(y_test, y_pred))
```

OUTPUT

```
[0.72226528 0.23974795 0.03338117 0.0046056 ]
[0.72226528 0.23974795 0.03338117 0.0046056 ]
[[11  0  0]
 [ 0 12  1]
 [ 0  1  5]]
Accuracy 0.9333333333333333
```

Learnings

The code first loads the Iris dataset, which is a collection of data about Iris flowers. The data includes four features: sepal length, sepal width, petal length, and petal width. The data also includes a label, which is the species of the Iris flower.

The code then splits the data into two sets: a training set and a test set. The training set is used to train the machine learning model, and the test set is used to evaluate the model's performance.

Before training the model, the code standardizes the features. This means that the code scales the features so that they are all on the same scale. This is important because it ensures that the model does not learn to bias towards any particular feature.

The code then uses PCA to reduce the dimensionality of the features. This means that the code combines the four features into a single feature. This can improve the model's performance because it reduces noise in the data and makes it easier for the model to learn the patterns.

The code then trains a Random Forest Classifier on the training set. A Random Forest Classifier is a type of machine learning model that is well-suited for classification tasks.

Once the model is trained, the code uses it to make predictions on the test set. The code then evaluates the model's performance using a confusion matrix and accuracy score. A confusion matrix shows how many flowers of each species were correctly and incorrectly classified by the model. An accuracy score is a measure of how often the model correctly classified a flower.

Practical No: 8

Least Square Regression Algorithm | Logistic Regression algorithm

AIM: For a given set of training data examples stored in a CSV. File implement Least Square Regression Algorithm.

Description:

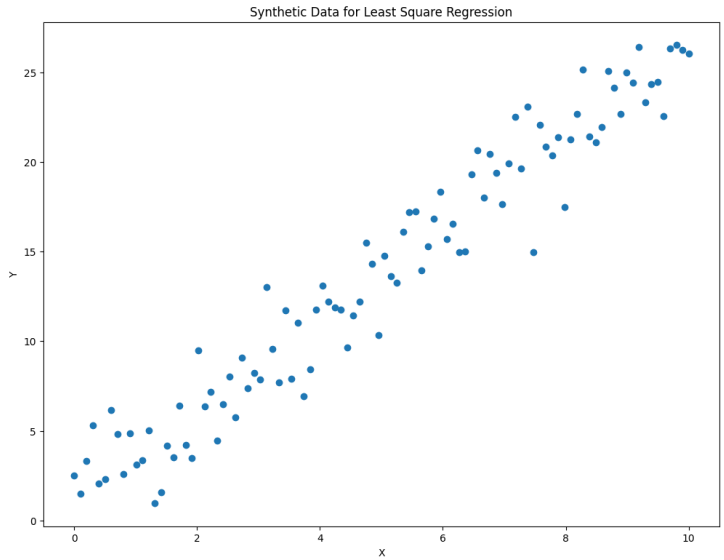
Least Square Regression is a linear regression algorithm used to find the best-fitting line through data points by minimizing the sum of squared differences between observed and predicted values. It calculates the slope and intercept that minimize the vertical distances of data points from the regression line, providing a straightforward method for modeling linear relationships.

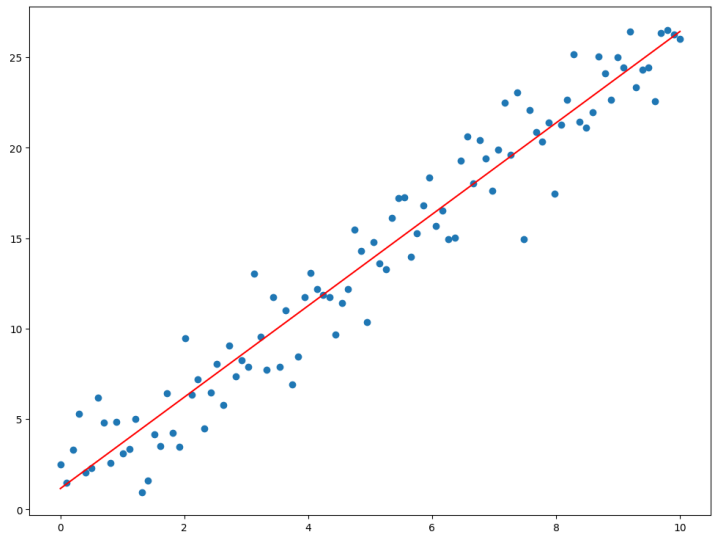
Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12.0, 9.0)

np.random.seed(42)
X = np.linspace(0, 10, 100)
Y = 2.5 * X + 1.5 + np.random.normal(0, 2, 100)
data = pd.DataFrame({"X": X, "Y": Y})
plt.scatter(X, Y)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Synthetic Data for Least Square Regression")
plt.show()
import numpy as np
# Assuming X and Y are your data arrays
X_mean = np.mean(X)
Y_mean = np.mean(Y)
num = 0
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean) * (Y[i] - Y_mean)
    den += (X[i] - X_mean) ** 2
m = num / den
c = Y_mean - m * X_mean
print(m, c)
# Making predictions
Y_pred = m * X + c
plt.scatter(X, Y)  # actual
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color="red")  # prediction
plt.show()
```

OUTPUT:





Learnings

The provided Python script aims to implement the Least Square Regression algorithm for a given set of training data stored in a CSV file. The synthetic data is generated, and a scatter plot is visualized to showcase the relationship between the independent variable (X) and the dependent variable (Y). The Least Square Regression is then applied manually, calculating the slope (m) and intercept (c) to define the best-fit line. Finally, predictions are made using these parameters, and both the actual data points and the regression line are plotted for visualization.

B) For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm

Code with output

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-
gurucharan/Classification/master/DMVWrittenTests.csv')
X = dataset.iloc[:, [0, 1]].values
Y = dataset.iloc[:, 2].values
dataset.head(5)
```

[1] ✓ 1.8s

	DMV_Test_1	DMV_Test_2	Results
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

```
# Splitting the dataset into the training set and test set.
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
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the logistic regression model on the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, Y_train)
```

```
# Predicting the test set results.
y_pred = classifier.predict(X_test)
y_pred
```

```
[3]  ✓ 0.0s

···  array([1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
        1, 1, 0], dtype=int64)
```

```
# Confusion Matrix and Accuracy.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_pred)
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(Y_test, y_pred))
cm
```

```
[4]  ✓ 0.0s

···  Accuracy: 0.88

···  array([[11,  0],
        [ 3, 11]], dtype=int64)
```

Learnings

This code implements a logistic regression model for classifying DMV written test results. It utilizes the scikit-learn library to preprocess the dataset, splitting it into training and testing sets. Feature scaling is applied for standardization, and the logistic regression model is trained on the training set. The predictions are then evaluated using a confusion matrix, providing insights into the model's performance. The accuracy score is calculated, representing the proportion of correctly classified instances. This code demonstrates a basic yet effective application of logistic regression for binary classification, essential in scenarios like DMV written test outcome prediction based on specific features.

Practical No: 9

Backpropagation algorithm and Text pre-processing, Text clustering, classification

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

Description:

Artificial Neural Network (ANN) with Backpropagation is a machine learning model designed to mimic the human brain's learning process. By iteratively adjusting connection weights during training, it learns intricate patterns in data, making it capable of making predictions and generalizing from the provided datasets.

Code and output:

```
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)
Y=Y/100;


class NN(object):
    def __init__(self):
        self.inputsize=2
        self.outputsize=1
        self.hiddensize=3
        self.W1=np.random.randn(self.inputsize,self.hiddensize)
        self.W2=np.random.randn(self.hiddensize,self.outputsize)
    def forward(self,X):
        self.z=np.dot(X,self.W1)
        self.z2=self.sigmoidal(self.z)
        self.z3=np.dot(self.z2,self.W2)
        op=self.sigmoidal(self.z3)
        return op;
    def sigmoidal(self,s):
        return 1/(1+np.exp(-s))
    def sigmoidalprime(self,s):
```

```
        return s* (1-s)
    def backward(self,X,Y,o):
        self.o_error=Y-o
        self.o_delta=self.o_error * self.sigmoidalprime(o)
        self.z2_error=self.o_delta.dot(self.W2.T)
        self.z2_delta=self.z2_error * self.sigmoidalprime(self.z2)
        self.W1 = self.W1 + X.T.dot(self.z2_delta)
        self.W2= self.W2+ self.z2.T.dot(self.o_delta)
    def train(self,X,Y):
        o=self.forward(X)
        self.backward(X,Y,o)
obj=NN()
for i in range(2000):
    print("input"+str(X))
    print("Actual output"+str(Y))
    print("Predicted output"+str(obj.forward(X)))
    print("loss"+str(np.mean(np.square(Y-obj.forward(X)))))
    obj.train(X,Y)
```

```
obj.train(X,Y)
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.32998547]
 [0.34536101]
 [0.34195292]]
loss0.30444200992247783
input[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.40142014]
 [0.41416845]
 [0.41504779]]
```

Learnings:

This code creates a basic computer program that tries to learn patterns from a small set of information. It uses a special math function to make predictions and adjusts itself to get better over 2000 tries. However, it needs some extra details, like how fast it should learn and a small fix to improve its performance. In a nutshell, it's like a beginner's attempt at building a smart program that needs a bit of fine-tuning.

b) AIM: Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix

DESCRIPTION:

1. Text Pre-processing:

Clean and prepare the restaurant review text data by removing non-alphabetic characters, converting to lowercase, stemming, and eliminating common English stopwords, ensuring the dataset is ready for analysis.

2. Text Clustering:

Utilize a Bag of Words model with CountVectorizer to transform the pre-processed text data into numerical features, enabling the application of clustering algorithms to group similar reviews together and identify patterns within the dataset.

3. Classification with Prediction:

Train a Gaussian Naive Bayes classifier on the pre-processed and transformed data to predict sentiment labels (positive or negative) for restaurant reviews, allowing the model to learn from the training set and make predictions on unseen data.

4. Test Score:

Evaluate the performance of the Naive Bayes classifier by calculating accuracy scores, using metrics such as accuracy_score to measure the model's effectiveness in correctly predicting sentiments on the test set.

5. Confusion Matrix:

Generate a confusion matrix to provide a detailed breakdown of the model's predictions, showcasing true positive, true negative, false positive, and false negative results. This matrix offers insights into the classifier's strengths and weaknesses in sentiment classification.

Code and output:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
 dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)
 import re
import nltk
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = []
for i in range(0,1000):
  review = re.sub('[^a-zA-Z]',",dataset['Review'][i])
  review = review.lower()
  review = review.split()
  ps = PorterStemmer()
  review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
  review = ".join(review)
  corpus.append(review)
#Creating the bag of words model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray()
Y = dataset.iloc[:,1].values
 #Splitting the dataset into the training set and test set
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=100)
 #Fitting naive bayes to the training set.
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
# Predicting the test set results.
Y_pred = classifier.predict(X_test)
 #Model Accuracy
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))
#Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
Out[3]: GaussianNB()
```

```
In [4]: # Predicting the test
        Y_pred = classifier.pr

        #Model Accuracy
        from sklearn import me
        from sklearn.metrics i
        print("Accuracy:",metr

        Accuracy: 0.54
```

```
In [5]: #Making the confusion
        from sklearn.metrics i
        cm = confusion_matrix(
        print(cm)

        [[  1 115]
         [  0 134]]
```

Learnings:

This code performs sentiment analysis on restaurant reviews using a Naive Bayes classifier. Initially, the dataset is loaded and pre-processed, including removing non-alphabetic characters, converting text to lowercase, and stemming words. The Bag of Words model is then implemented using the Count Vectorizer to transform the text data into numerical features. The dataset is split into training and testing sets, with 75% for training and 25% for testing. A Gaussian Naive Bayes classifier is trained on the training set and used to predict sentiments on the test set. The model's accuracy is evaluated using metrics like accuracy score and a confusion matrix, providing insights into the classifier's performance. This approach leverages natural language processing techniques, including text cleaning and machine learning, to classify reviews as positive or negative based on the words used. The Naive Bayes model proves useful for its simplicity and effectiveness in handling textual data.

Practical No: 10

Distance methods with Prediction | K – Means Clustering.

10 A) Aim: Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

Code and Output:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
#Load the dataset
df = pd.read_csv("../dataset/Iris.csv")

#quick look into the data
print(df.head(5))
print("\n")
#Separate data and label
x = df.drop(['Species'], axis=1)
y = df['Species']
#Prepare data for classification process
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
#Create a model , p = 2 => Euclidean Distance:
knn = KNeighborsClassifier(n_neighbors = 6, p = 2, metric='minkowski')
#Train the model
knn.fit(x_train, y_train)
# Calculate the accuracy of the model
print("Accuracy of Euclidean Distance model:-")
print(knn.score(x_test, y_test))
y_pred = knn.predict(x_test)
#confusion matrix
```

```
from sklearn.metrics import  confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
print("-"*50)
print("Confusion matrix:-")
print(cm)
print("\n")
#Create a model , p = 1 => Manhattan Distance
knn = KNeighborsClassifier(n_neighbors = 6, p = 1, metric='minkowski')
#Train the model
knn.fit(x_train, y_train)
# Calculate the accuracy of the model
print("-"*50)
print("Accuracy of Manhattan Distance model:-")
print(knn.score(x_test, y_test))
y_pred = knn.predict(x_test)
#confusion matrix
from sklearn.metrics import  confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
print("-"*50)
print("Confusion matrix:-")
print(cm)
print("\n")
#Create a model ,p = ∞, Chebychev Distance
#let ∞ = 10000
knn = KNeighborsClassifier(n_neighbors = 6, p = 10000, metric='minkowski')

#Train the model
knn.fit(x_train, y_train)
# Calculate the accuracy of the model
print("-"*50)
print("Accuracy of Chebychev Distance model:-")
print(knn.score(x_test, y_test))
y_pred = knn.predict(x_test)
#confusion matrix
```

```
from sklearn.metrics import  confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
print("-"*50)
print("Confusion matrix:-")
print(cm)
print("\n")
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0            5.1           3.5            1.4           0.2  Iris-setosa
1            4.9           3.0            1.4           0.2  Iris-setosa
2            4.7           3.2            1.3           0.2  Iris-setosa
3            4.6           3.1            1.5           0.2  Iris-setosa
4            5.0           3.6            1.4           0.2  Iris-setosa
```

```
Accuracy of Euclidean Distance model:-
0.9777777777777777
--------------------------------------------------
Confusion matrix:-
[[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
```

```
--------------------------------------------------
Accuracy of Manhattan Distance model:-
0.9555555555555556
--------------------------------------------------
Confusion matrix:-
[[16  0  0]
 [ 0 17  1]
 [ 0  1 10]]
```

```
--------------------------------------------------
Accuracy of Chebychev Distance model:-
0.8
--------------------------------------------------
Confusion matrix:-
[[16  0  0]
 [ 0 18  0]
 [ 0  9  2]]
```

10B: AIM: Implement the classification model using K-means clustering with Prediction, Test score and Confusion Matrix.

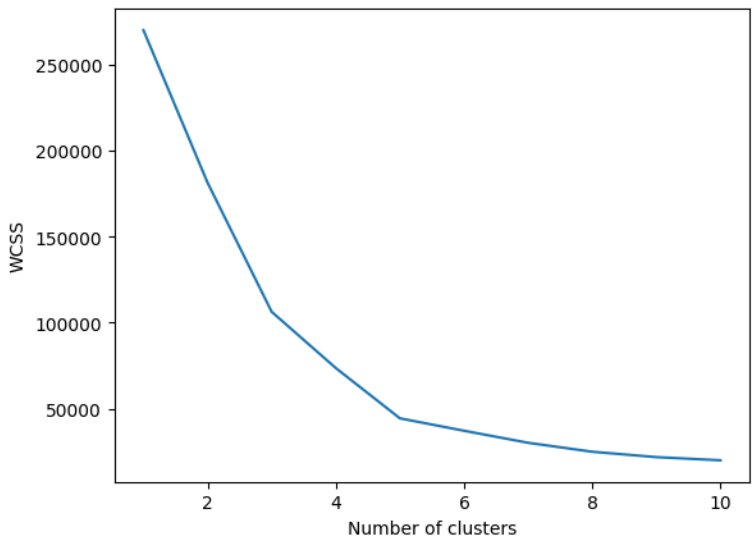
Description:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

Code and output:

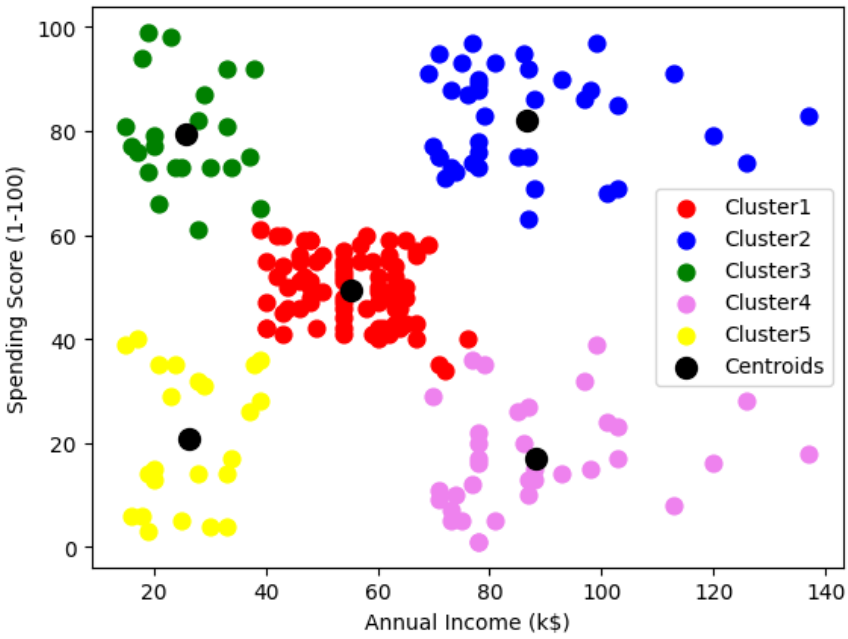
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
#Import the dataset and slice the important features
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3,4]].values
#Find the optimal k value for clustering the data.
from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```
#The point at which the elbow shape is created is 5.
kmeans = KMeans(n_clusters=5,init="k-means++",random_state=42)
y_kmeans = kmeans.fit_predict(X)

plt.scatter(X[y_kmeans == 0,0], X[y_kmeans == 0,1], s = 60, c = 'red', label = 'Cluster1')
plt.scatter(X[y_kmeans == 1,0], X[y_kmeans == 1,1], s = 60, c = 'blue', label = 'Cluster2')
plt.scatter(X[y_kmeans == 2,0], X[y_kmeans == 2,1], s = 60, c = 'green', label = 'Cluster3')
plt.scatter(X[y_kmeans == 3,0], X[y_kmeans == 3,1], s = 60, c = 'violet', label = 'Cluster4')
plt.scatter(X[y_kmeans == 4,0], X[y_kmeans == 4,1], s = 60, c = 'yellow', label = 'Cluster5')
plt.scatter(kmeans.cluster_centers_[:,0],
kmeans.cluster_centers_[:,1],s=100,c='black',label='Centroids')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100')
plt.legend()
plt.show()
```



Learning:

This code snippet demonstrates the implementation of K-Means clustering on a Mall Customers dataset using Python's scikit-learn library. It first imports necessary modules and reads the dataset, selecting two key features – Annual Income and Spending Score. The optimal number of clusters (k) is determined by plotting the Within-Cluster-Sum-of-Squares (WCSS) against different k values. In this case, the elbow method suggests k=5. The K-Means algorithm is then applied, and the clusters are visualized with a scatter plot, showcasing distinct clusters based on customers' Annual Income and Spending Score. The black points represent cluster centroids, providing insights into customer segmentation for targeted business strategies.