GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University)

Coimbatore - 641 013

DEPARTMENT OF CSE

2018 REGULATIONS

18SPC707 – MACHINE LEARNING LABORATORY

18SPC707	MACHINE LEARNING LABORATORY	SEMESTER VII

PRE-REQUISITES: NIL **Category: PC**

L	T	P	C
0	0	4	2

COURSE OBJECTIVES:

Upon completion of this course, the students will be familiar with,

- Basic knowledge of Machine learning tools.
- Basic concepts of learning algorithm.
- Decision Tree and Support Vector Machines.
- Back-propagation algorithm and Naive Bayes Algorithm.
- Instant based learning algorithm and Clustering
- Hidden Markov Models.

LIST OF EXPERIMENTS

EXERCISES ILLUSTRATING THE FOLLOWING CONCEPTS:

1. Study of machine learning tool R.
2. Study of machine learning tool Scikit Learn.
3. Implement the CANDIDATE – ELIMINATION algorithm. Show how it is used to learn from training examples and hypothesize new instances in Version Space.
4. Implement the FIND–S algorithm. Show how it can be used to classify new instances of target concepts. Run the experiments to deduce instances and hypothesis consistently.
5. Implement the ID3 algorithm for learning Boolean–valued functions for classifying the training examples by searching through the space of a Decision Tree.
6. Design and implement the Back-propagation algorithm by applying it to a learning task involving an application like FACE RECOGNITION.
7. Design and implement Naive Bayes Algorithm for learning and classifying TEXTDOCUMENTS.
8. Design and implement K-nearest Neighbor learning algorithm.
9. Design and implement Support Vector Machines.
10. Design and implement the Hidden Markov Models.
11. Design and implement the Clustering algorithm.

Contact Periods:

Lecture: 0 Periods **Tutorial: 0 Periods** **Practical: 60 Periods** **Total: 60 Periods**

COURSE OUTCOMES:

Upon completion of this course, the students will be able to,

CO1: Use appropriate Machine Learning tools **[Usage]**
CO2: Implement the CANDIDATE – ELIMINATION algorithm **[Usage]**
CO3: Implement the FIND–S algorithm **[Usage]**
CO4: Implement the ID3 algorithm.**[Usage]**
CO5: Design and implement the Back-propagation algorithm **[Usage]**
CO6: Design and implement Naïve Bayes Algorithm **[Usage]**
CO7: Design and implement K-nearest Neighbor learning algorithm. **[Usage]**
CO8: Design and implement Support Vector Machines. **[Usage]**
CO9. Design and implement the Hidden Markov Models. **[Usage]**
CO10. Design and implement the Clustering algorithm. **[Usage]**

COURSE ARTICULATION MATRIX:

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3	PSO 4
CO1	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO2	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO3	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO4	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO5	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO6	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO7	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO8	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO9	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
CO10	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L
18SPC 707	H	H	H	M	H	L	M	H	H	H	H	M	H	M	H	L

L - Low, M - Moderate (Medium), H - High

INDEX

EXP: 1 STUDY OF MACHINE LEARNING TOOL-R

AIM:

To study about the Machine Learning tool R .

DESCRIPTION:

Download and install R and get the most useful package for machine learning in R. Load a dataset and understand it's structure using statistical summaries and data visualization. Create various machine learning models, pick the best and build confidence that the accuracy is reliable.

Why R is used than rest of the languages?

R was designed to do data analysis. So it is heavy on graphics, statistical functionality, data structures, object manipulation, and numerical computation. R is an interpreted language that is vector-oriented, and has lexical scoping. Influences include functional programming and object-orientation.

How R is used in ML?

The R platform is not suitable for all types of machine learning projects. The sweet spot is to use R for exploration and for building one-off models. The R interactive environment is very useful for exploring and learning how to use packages and functions. You should spend a lot of time in the interactive environment when you are just starting out. The environment is also very good if you are exploring a new problem. Not a systematic working of the problem, but more of trying what-if scenarios. It is also great if you want to use a systematic process and come up with a prototype model very quickly without the full rigmarole.

Steps for Installation:

1) Download the installable file from the following link:
   https://cran.r-project.org/bin/windows/base/
2) Click on the R 3.2.2.exe file. The 3.2.2 is the version number of the file.
   The versions can be updated as per the latest releases.
3) The SetUp will request permission to be installed on the system click yes to proceed.
4) Select the Preferred language from the dropdown to begin an installation in that preferred language
5) Click next to proceed with the installation.
6) Choose the path where you wish to install R by clicking on browse and changing the workspace locations. Click next to proceed with the default installation. The minimum space

requirements are mentioned at the bottom of the dialog box. Please check you have required amount of free space in your drive.

7) Choose the type of installation you require. By default R installs both the 32 and 64 bit versions on your system. If your system is a 32 bit system you will be requiring a 32 bit installation if the system is a 64 bit system it will be requiring 64 bit installation. Do not uncheck the Core Files and Message Translations. Please make note of the space requirement of the installation.

8) To customize the start-up options for R choose option and customize. To proceed with a vanilla installation use Next.

9) To generate program shortcuts and naming those as per your requirement specify the necessary customizations. To proceed with the default installation hit next.

10) Click on the next button to begin your installation.

11) After the installation has completed you will see the final screen. Click finish to complete the installation.

12) Open Start Menu and you will find R in the available set of Programs.

Step 1-Installation of Packages:

The caret package provides a consistent interface into hundreds of machine learning algorithms and provides useful convenience methods for data visualization, data resampling, model tuning and model comparison, among other features. It's a must have tool for machine learning projects in R.

```
install.packages("caret")
install.packages("caret",dependencies=c("Depends","Suggests"))
 library(caret)
```

Step 2-Load the data set:

Iris flower dataset is used here. The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimetres. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

Load the data:

```
data(iris)
dataset<-iris
```

Load from CSV:

```
filename<-"iris.csv"
```

```
dataset<-read.csv(filename,header=FALSE)
colnames(dataset)<-c("Sepal.Length","Sepal.With","Petal.Length","Petal.Width","Species")
```

Create a validation dataset:

```
validation_index<-createDataPartition(dataset$Species,p=0.80,list=FALSE)
validation<-dataset[-validation_index,]
dataset<-dataset[validation_index,]
```

Step 3-Summarize Dataset:

Dimensions of the dataset.

```
dim(dataset)
```

Types of the attributes.

```
sapply(dataset,class)
```

Peek at the data itself.

```
head(dataset)
```

Class Distribution

```
percentage <- prop.table(table(dataset$Species)) * 100
cbind(freq=table(dataset$Species), percentage=percentage)
```

Statistical summary

```
summary(dataset)
```

Step 4-Visualize Dataset:

Univariate plots helpful with visualization to have a way to refer to just the input attributes and just the output attributes.

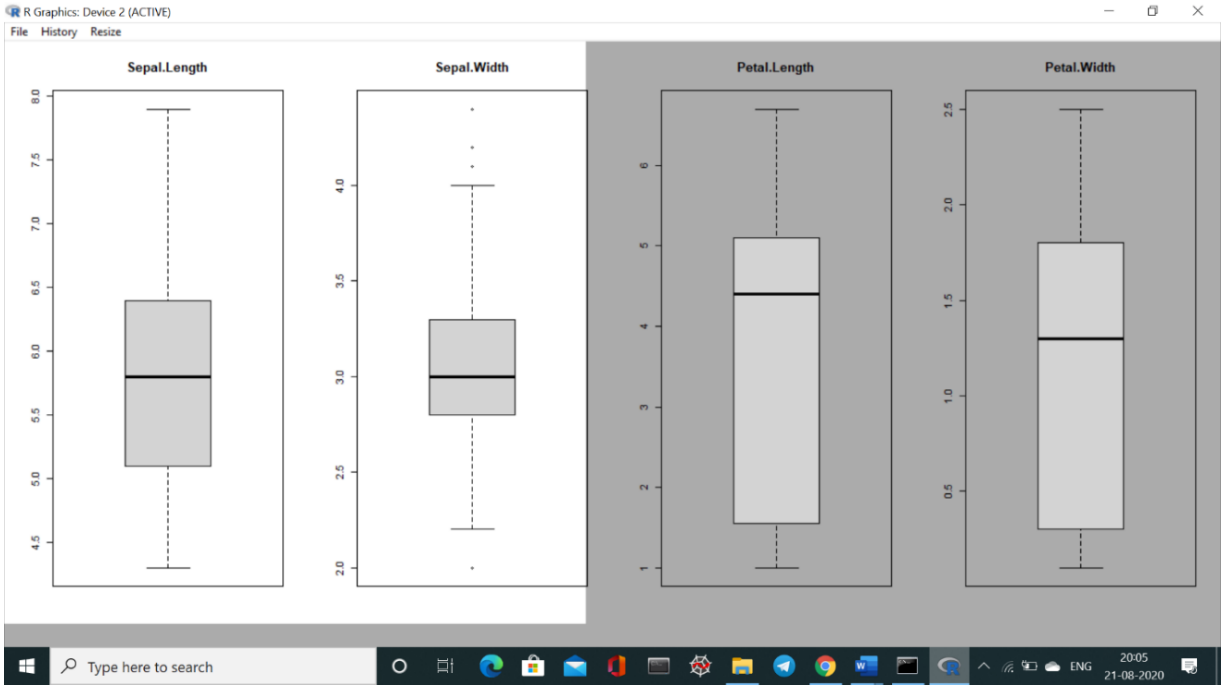
Step 5-Splitting Dataset:

```
x <- dataset[,1:4]
y <- dataset[,5]
```

Visualising:

```
par(mfrow=c(1,4))
  for(i in 1:4)
  {
 boxplot(x[,i], main=names(iris)[i])
    }
```

Sample output for Visualisation of Iris dataset:



Result:

Thus the study of Machine Learning tool R is done successfully.

EXP: 2 STUDY OF MACHINE LEARNING TOOL -SCIKIT

Aim:

To study the machine learning tool Scikit.

Dataset Loading :

A collection of data is called dataset. It is having the following two components −

Features− The variables of data are called its features. They are also known as predictors, inputs or attributes.

- **Feature matrix** − It is the collection of features, in case there are more than one.
- **Feature Names** − It is the list of all the names of the features.

Response− It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

- **Response Vector** − It is used to represent response column. Generally, we have just one response column.
- **Target Names** − It represent the possible values taken by a response vector.

Scikit-learn have few example datasets like **iris** and **digits** for classification and the **Boston house prices** for regression.

Example to load iris dataset −:

```
from sklearn. datasets
import load_iris iris =load_iris()
X =iris.data
y =iris.target
 from sklearn.model_selection
import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.3,random_state=1
print(X_train.shape)print(X_test.shape)
print(y_train.shape)print(y_test.shape)
```

Output :

Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal

length (cm)', 'petal width (cm)']

Target names: ['setosa' 'versicolor' 'virginica'] First 10 rows of X:
[
[5.1 3.5 1.4 0.2] [4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5.3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3] [5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

]

Splitting the dataset:

To check the accuracy of our model, we can split the dataset into two pieces-**a training set** and **a testing set**. Use the training set to train the model and testing set to test the model. After that, we can evaluate how well our model did.

Example to split the dataset

The following example will split the data into 70:30 ratio, i.e. 70% data will be used as training data and 30% will be used as testing data. The dataset is iris dataset as in above example.

```
from sklearn. Datasets
 import load_iris iris =load_iris()
X =iris.data
y =iris.target
from sklearn.model_selection
```

```
import train_test_split
X_train,X_test,y_train,y_test=train_test_split(
X, y,test_size=0.3,random_state=1
print(X_train.shape)print(X_test.shape)
print(y_train.shape)print(y_test.shape)
```

Output

```
(105, 4)

(45, 4)

(105,)

(45,)
```

As seen in the example above, it uses **train_test_split()** function of scikit-learn to split the dataset. This function has the following arguments −

- **X, y** − Here, **X** is the **feature matrix** and y is the **response vector**, which need to be split.
- **test_size** − This represents the ratio of test data to the total given data. As in the above example, we are setting **test_data = 0.3** for 150 rows of X. It will produce test data of 150*0.3 = 45 rows.
- **random_size** − It is used to guarantee that the split will always be the same. This is useful in the situations where you want reproducible results.

Train the Model:

Next, we can use our dataset to train some prediction-model. As discussed, scikit-learn has wide range of Machine Learning (ML) algorithms which have a consistent interface for fitting, predicting accuracy, recall etc.

Example to train the model:

In the example below, we are going to use KNN (K nearest neighbors) classifier. Don't go into the details of KNN algorithms, as there will be a separate chapter for that. This example is used to make you understand the implementation part only.

```
from sklearn. Datasets
 import load_iris iris =load_iris()
```

```
X =iris.data
y =iris.target
from sklearn.model_selection
 import train_test_splitX_train,X_test,y_train,y_test=train_test_split ( X,
y,test_size=0.4,random_state=1)
from sklearn. Neighbors
 import KNeighborsClassifier from sklearn
 import metrics classifier_knn=KNeighborsClassifier(n_neighbors=3)
classifier_knn.fit(X_train,y_train)y_pred=classifier_knn.predict(X_test)
# Finding accuracy by comparing actual response values(y_test) with predicted response
value(y_pred)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
# Providing sample data and the model will make prediction out of that
data sample =[[5,5,3,2],[2,4,3,5]]
preds =classifier_knn.predict(sample)
pred_species=[iris.target_names[p]
for p in preds]print("Predictions:",pred_species)
```

Output :

```
Accuracy: 0.9833333333333333

Predictions: ['versicolor', 'virginica']
```

Model Persistence :

Once you train the model, it is desirable that the model should be persist for future use so that we do not need to retrain it again and again. It can be done with the help of **dump** and **load** features of ***joblib*** package.

Consider the example below in which we will be saving the above trained model

(classifier_knn) for future use −

```
from sklearn.externals
import  joblib
joblib.dump(classifier_knn, 'iris_classifier_knn.joblib')
```

The above code will save the model into file named iris_classifier_knn.joblib. Now, the object can

be reloaded from the file with the help of following code −

```
joblib.load('iris_classifier_knn.joblib')
```

Pre-processing the Data:

As we are dealing with lots of data and that data is in raw form, before inputting that data to machine learning algorithms, we need to convert it into meaningful data. This process is called pre-processing the data. Scikit-learn has package named **pre-processing** for this purpose. The **pre-processing** package has the following techniques −

Binarisation :

This preprocessing technique is used when we need to convert our numerical values into Boolean values.

Example :

```
import numpy as np
From sklearn
 import preprocessing
Input_data = np.array(
[2.1,-1.9,5.5],
[-1.5,2.4,3.5],
[0.5,-7.9,5.6],
[5.9,2.3,-5.8]])
data_binarized=preprocessing.Binarizer(threshold=0.5).transform(input_data)
print("\nBinarized data:\n",data_binarized)
```

In the above example, we used **threshold value** = 0.5 and that is why, all the values above 0.5 would be converted to 1, and all the values below 0.5 would be converted to 0.

Output :

Binarized data:

```
[
  [ 1. 0. 1.]
  [ 0. 1. 1.]
```

[0. 0. 1.]

[1. 1. 0.]

]

Mean Removal:

This technique is used to eliminate the mean from feature vector so that every feature centered on zero.

Example:

```
import numpy as np
from sklearn import preprocessing
Input_data=np.array(
[2.1,-1.9,5.5],
[-1.5,2.4,3.5],
[0.5,-7.9,5.6],
[5.9,2.3,-5.8]]
)#displaying the mean and the standard deviation of the input dataprint
("Mean =",input_data.mean(axis=0))print("Stddeviation = ",input_data.std(axis=0))
#Removing the mean and the standard deviation of the input data
data_scaled=preprocessing.scale(input_data)
print("Mean_removed =",data_scaled.mean(axis=0))
print("Stddeviation_removed =",data_scaled.std(axis=0))
```

Output :

Mean = [1.75 -1.275 2.2]

Std_deviation = [2.71431391 4.20022321 4.69414529]

Mean_removed = [1.11022302e-16 0.00000000e+00 0.00000000e+00]

Std_deviation_removed = [1. 1. 1.]

Scaling :

We use this pre-processing technique for scaling the feature vectors. Scaling of feature vectors is important, because the features should not be synthetically large or small.

Example :

```
import numpy as np
from sklearn import preprocessing
Input_data=np.array(
[
[2.1,-1.9,5.5],
[-1.5,2.4,3.5],
[0.5,-7.9,5.6],
[5.9,2.3,-5.8]
])
data_scaler_minmax= preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax=data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n",data_scaled_minmax)
```

Output :

```
Min max scaled data:
[
  [ 0.48648649 0.58252427 0.99122807]
  [ 0. 1. 0.81578947]
  [ 0.27027027 0. 1. ]
  [ 1. 0.99029126 0. ]
]
```

Normalisation :

We use this pre-processing technique for modifying the feature vectors. Normalisation of feature vectors is necessary so that the feature vectors can be measured at common scale. There are two types of normalisation as follows

L1 Normalisation :

It is also called Least Absolute Deviations. It modifies the value in such a manner that the sum of the absolute values remains always up to 1 in each row. Following example shows the

implementation of L1 normalisation on input data.

Example :

```
import numpy as np
from sklearn import pre-processing
Input_data=np.array(
[
[2.1,-1.9,5.5],
[-1.5,2.4,3.5],
[0.5,-7.9,5.6],
[5.9,2.3,-5.8]
])
data_normalized_l1 =preprocessing.normalize(input_data, norm='l1')
print("\nL1 normalized data:\n", data_normalized_l1)
```

Output :

L1 normalized data:

```
[
  [ 0.22105263 -0.2 0.57894737]
  [-0.2027027 0.32432432 0.47297297]
  [ 0.03571429 -0.56428571 0.4 ]
  [ 0.42142857 0.16428571 -0.41428571] ]
```

L2 Normalisation :

Also called Least Squares. It modifies the value in such a manner that the sum of the squares remains always up to 1 in each row. Following example shows the implementation of L2 normalisation on input data.

Example :

```
import numpy as np

from sklearn import preprocessing

Input_data=np.array(

[

[2.1,-1.9,5.5],

[-1.5,2.4,3.5],

[0.5,-7.9,5.6],

[5.9,2.3,-5.8]

]

)

data_normalized_l2 =preprocessing.normalize(input_data, norm='l2')print("\nL1 normalized data:\n", data_normalized_l2)
```

Output:

L2 normalized data:

```
[ [ 0.33946114 -0.30713151 0.88906489] [-0.33325106 0.53320169 0.7775858]

  [ 0.05156558 -0.81473612 0.57753446]

  [ 0.68706914 0.26784051 -0.6754239]

]
```

Result:

Thus the study of machine learning tool scikit is done successfully.

EXP: 3 IMPLEMENTATION OF CANDIDATE ELIMINATION ALGORITHM

Aim:

To implement the candidate elimination algorithm and show how it is used to learn from training examples and hypothesize new instances in version space.

Description:

o Consider both positive and negative result. oFor positive example: tend to generalize specific hypothesis.

o For Negative example: tend to make general hypothesis more specific oWill use version space.

Algorithm:

1. Load the dataset.

2. Initialize G to the set of maximally general hypotheses in H

3. Initialize S to the set of maximally specific hypotheses

For 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

CODE :

```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_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)
    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                 if target[i] == "no":
            print("Instance is Negative ")
            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("Specific Boundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")
    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

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']

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', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]


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', '?', '?', '?', '?']]
```

RESULT:

Thus the candidate elimination algorithm has been implemented successfully.

EXP: 4 IMPLEMENTATION OF FIND-S ALGORITHM

Aim:

To implement the find-s algorithm to classify new instances of target concepts.

Find-S Algorithm in Machine Learning:

- Concept Learning
- General Hypothesis
- Specific Hypothesis

1. Concept Learning

In Machine Learning, concept learning can be termed as "*a problem of searching through a predefined space of potential hypothesis for the hypothesis that best fits the training examples".*

Any algorithm that supports concept learning requires the following:

- Training Data
- Target Concept
- Actual Data Objects

2. General Hypothesis

The general hypothesis basically states the general relationship between the major variables. For example, a general hypothesis for ordering food would be I want a burger.

G = { '?', '?', '?', …..'?'}

3. Specific Hypothesis

The specific hypothesis fills in all the important details about the variables given in the general hypothesis. The more specific details into the example given above would be I want a cheeseburger with a chicken pepperoni filling with a lot of lettuce.

S = {'Φ','Φ','Φ', ……,'Φ'}

Algorithm:

The Find-S algorithm follows the steps written below:

1. The process starts with initializing ‘h’ with the most specific hypothesis, generally, it is the first positive example in the data set.
2. We check for each positive example. If the example is negative, we will move on to the next example but if it is a positive example we will consider it for the next step.
3. We will check if each attribute in the example is equal to the hypothesis value.
4. If the value matches, then no changes are made.
5. If the value does not match, the value is changed to ‘?’.
6. We do this until we reach the last positive example in the data set.

Implementation of Find-S Algorithm:

DATASET:

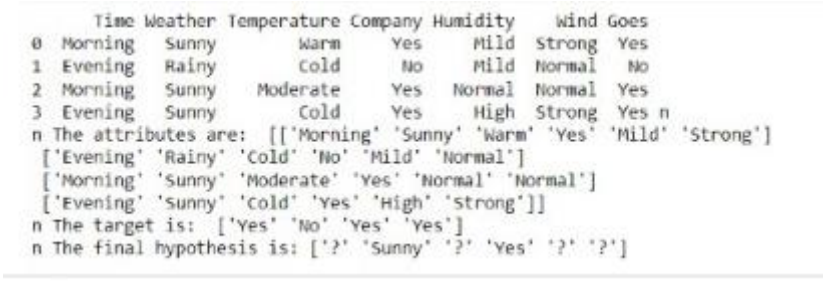
Weather	Temperature	Humidity	Wind	Temperature	Climate	Goes
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Cool	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

CODE:

```
import pandas as pd
import numpy as np
 #to read the data in the csv file
data = pd.read_csv("weatther_data.csv")
print(data,"n")
 #making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)
 #segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)
```

```
#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
                for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else: pass
                    return specific_hypothesis
#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

OUTPUT:



```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes n
n The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'strong']]
n The target is:  ['Yes' 'No' 'Yes' 'Yes']
n The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

RESULT:

Thus the find-s algorithm has been implemented successfully.

EXP: 5 IMPLEMENTATION OF ID3 ALGORITHM

Aim:

To implement the ID3 algorithm for classifying the training examples by searching through the space of a decision tree.

Algorithm:

1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,

   Return the single-node tree Root, with label = most common value of Target Attribute in Examples

   Else

   A ← the attribute from Attributes that best classifies Examples

   The decision attribute for Root ←A

   For each possible value, vi, of A,

   Add a new tree branch below Root, corresponding to the test A = vi

   Let Examples be the subset of Examples that have value vi for A
5. If Examples is empty

   Then below this new branch add a leaf node with label = most common value of Target Attribute in Examples

   Else

   Then below this new branch add the subtree ID3(Examples, Target Attribute, Attributes–{A})

   End
6. Return Root

CODE:

```
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))
for k in attr:
dic[k] = []
for y in range(len(data)):
 key = data[y][col]
if delete:
    del data[y][col]
 dic[key].append(data[y])
 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):
 attValues, dic = subtables(data, col, delete=False)
total_entropy = entropy([row[-1] for row in data])
for x in range(len(attValues)):
 ratio = len(dic[attValues[x]]) / ( len(data) * 1.0)
entro = entropy([row[-1]
for row in dic[attValues[x]]])
total_entropy -= ratio*entro
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 = [compute_gain(data, col) for col in range(n) ]
split = gains.index(max(gains))
node = Node(features[split])
 #del (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")
node = build_tree(dataset, features)
print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
 print("The test instance : ",xtest)
print("The predicted label : ", end="")
classify(node,xtest,features)
```

Dataset:

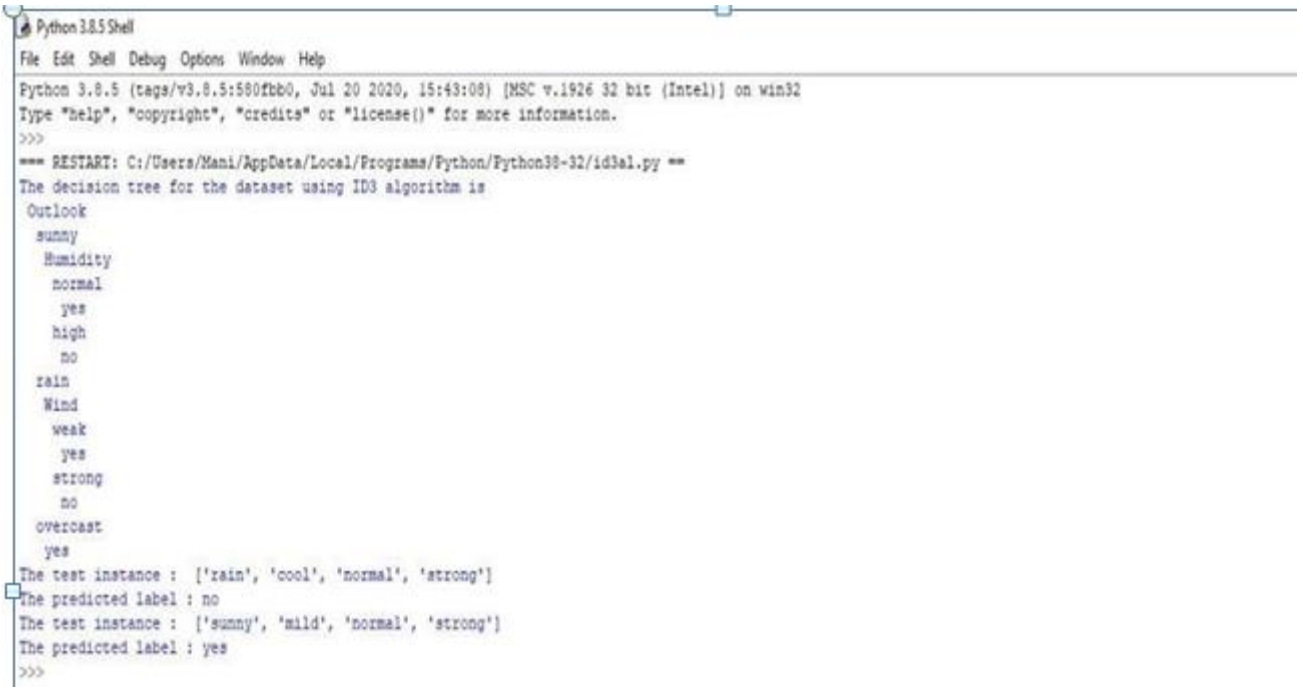
Training instances: data3.csv

Outlook, Temperature, Humidity, Wind, Target

Testing instances: data3_test.csv

Outlook, Temperature, Humidity, Wind, rain, cool, normal, strong sunny mild, normal, strong

OUTPUT:



\

RESULT:

Thus the ID3 algorithm has been implemented successfully.

EXP: 6 IMPLEMENTATION OF BACK PROPAGATION ALGORITHM

AIM:

To implement the Back Propagation Algorithm and test the same using appropriate data sets.

DESCRIPTION:

Back propagation is a supervised learning algorithm. The Back propagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

ALGORITHM:

1. Load data set
2.Assign all network inputs and output
3.Initialize all weights with small random numbers, typically between -1 and 1 repeat for every pattern in the training set
4.Present the pattern to the network
// Propagated the input forward through the network:
5.For each layer in the network for every node in the layer
    1. Calculate the weight sum of the inputs to the node
    2. Add the threshold to the sum
    3. Calculate the activation for the node end

end
// Propagate the errors backward through the network for every node in the output layer calculate the error signal end
6.For all hidden layers for every node in the layer
    1.Calculate the node's signal error
    2.Update each node's weight in the network end

end
// Calculate Global Error

7.Calculate the Error

Function end while ((maximum number of iterations < than specified) AND(Error Function is > than specified))

CODE:

facepickle.py:

```
from imutils import paths
import face_recognition
import pickle
import cv2
import os
 imagePaths =list(paths.list_images("/home/mariappan/Documents/mari/project/DatasetFinal"))
knownEncodings = []
 knownNames = []
for (i, imagePath) in enumerate(imagePaths):
print("processing image "+str(i)) name = imagePath.split(os.path.sep)[-2]
image = cv2.imread(imagePath)
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
boxes = face_recognition.face_locations(rgb,model="cnn")
encodings = face_recognition.face_encodings(rgb, boxes)
for encoding in encodings:
knownEncodings.append(encoding) knownNames.append(name)
data = {"encodings": knownEncodings, "names": knownNames}
f = open("/home/mariappan/Documents/mari/project/osus.pickle", "wb")
f.write(pickle.dumps(data))
f.close()
```

image.py:

```
import face_recognition
import pickle import cv2
data = pickle.loads(open("/home/mariappan/Documents/mari/project/faces.pickle",
"rb").read())
image = cv2.imread("/home/mariappan/Documents/mari/project/mari.jpg")
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
boxes = face_recognition.face_locations(rgb,model="cnn")
encodings = face_recognition.face_encodings(rgb, boxes)
names = [] for encoding in encodings:
matches = face_recognition.compare_faces(data["encodings"],encoding)
name = "Unknown"
if True in matches:
matchedIdxs = [i for (i, b) in enumerate(matches) if b]
counts = {}
for i in matchedIdxs:
```

```
name = data["names"][i]
counts[name] = counts.get(name, 0) + 1 name = max(counts, key=counts.get)
print(str(counts[name])+name)
names.append(name)
for ((top, right, bottom, left), name) in zip(boxes, names):
cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2) y = top - 15 if top - 15 > 15
else
top + 15
cv2.putText(image, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
0.75, (0, 255, 0), 2)
cv2.imshow("Image", image) cv2.waitKey(0)
```

webcam.py:

```
from imutils.video
import VideoStream
import face_recognition
import imutils
import pickle
import time
import cv2
data = pickle.loads(open("/home/mariappan/Documents/mari/project/faces.pickle",
"rb").read())
vs = VideoStream(src=0).start() writer = None time.sleep(2.0) while True:
frame = vs.read()
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
rgb = imutils.resize(frame, width=750) r = frame.shape[1] / float(rgb.shape[1])
boxes = face_recognition.face_locations(rgb,model="cnn")
encodings = face_recognition.face_encodings(rgb, boxes)
names = []
for encoding in encodings:
matches = face_recognition.compare_faces(data["encodings"],encoding)
name = "Unknown"
if True in matches:
matchedIdxs = [i for (i, b) in enumerate(matches) if b]
counts = {}
for i in matchedIdxs:
name = data["names"][i]
counts[name] = counts.get(name, 0) + 1
name = max(counts, key=counts.get)
print(str(counts[name])+name)
names.append(name)
for ((top, right, bottom, left), name) in zip(boxes, names):
top = int(top * r) right = int(right * r)
bottom = int(bottom * r)
left = int(left * r)
cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
y = top - 15 if top - 15 > 15
else
top + 15
cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
```

```
0.75, (0, 255, 0), 2)
if writer is None: fourcc = cv2.VideoWriter_fourcc(*"MJPG")
writer = cv2.VideoWriter("/home/mariappan/Documents/mari/project/faces.pickle/mari1.avi",
fourcc, 20, (frame.shape[1], frame.shape[0]), True)
writer. write(frame) cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF if key == ord("q"): break
cv2.destroyAllWindows()
vs.stop()
if writer is not None:
writer. release()
```

video.py:

```
import face_ recognition
import imutils
import pickle
import time
import cv2
data = pickle.loads(open("/home/mariappan/Documents/mari/project/faces.pickle",
"rb").read())
stream = cv2.VideoCapture("/home/mariappan/Documents/mari/project/mari_input.mp4")
writer = None while True:
(grabbed, frame) = stream.read()
if not grabbed:
break
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
rgb = imutils.resize(frame, width=750) r = frame.shape[1] / float(rgb.shape[1])
boxes = face_recognition.face_locations(rgb,model="cnn")
encodings = face_recognition.face_encodings(rgb, boxes)
names = [] for encoding in encodings:
matches = face_recognition.compare_faces(data["encodings"],encoding)
name = "Unknown"
if True in matches:
matchedIdxs = [i for (i, b) in enumerate(matches) if b]
counts = {}
for i in matchedIdxs:
 name = data["names"][i]
counts[name] = counts.get(name, 0) + 1
name = max(counts, key=counts.get)
names.append(name)
for ((top, right, bottom, left), name) in zip(boxes, names):
top = int(top * r)
right = int(right * r)
bottom = int(bottom * r)
left = int(left * r)
cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
y = top - 15 if top - 15 > 15
else
top + 15
```

```
cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
if writer is None:
fourcc = cv2.VideoWriter_fourcc(*"MJPG")
writer = cv2.VideoWriter("/home/mariappan/Documents/mari/project/mari_output.avi", fourcc, 24,
(frame. shape[1], frame. shape[0]), True)
writer. write(frame)
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
if key == ord("q"):
break
stream. release()
if writer is not None:
writer. release()
```

OUTPUT:

Result:

Thus, the implementation of back propagation algorithm is executed successfully.

EXP: 7 DESIGN AND IMPLEMENTATION OF NAIVE BAYES ALGORITHM FOR LEARNING AND CLASSIFYING TEXT DOCUMENTS

Aim:

To assume a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Algorithm:

1. Given training data set D which consists of documents belonging to different class say class A and B.
2. Calculate the prior probability of class A= number of objects of class A/total number of objects.

   Calculate the prior probability of class B= number of objects of class B /total number of objects.
3. Find ni, the total number of word frequency of each class.

   na= the total number of word frequency of class A.

   nb= the total number of word frequency of class B .

4.Find conditional probability of keyword occurrence given a class

P(word1 / class A)=wordcount / ni(A)

P(word1 / class B)=wordcount / ni(B)

P(word2 / class A)=wordcount / ni(A)

P(word2 / class B)=wordcount / ni(B)

…….

…….

P(wordn / class B)=wordcount / ni(B)

5.Avoid zero frequency problems by applying uniform distribution.

6.Classify a new document C based on the probability P(C/W)

a).Find P(A/W)=P(A)*P(word1 /class A)* P(word2 /class A)…….*P(wordn /class A)

b). Find P(B/W)=P(B)*P(word1 /class B)* P(word2 /class B)…….*P(wordn /class B)

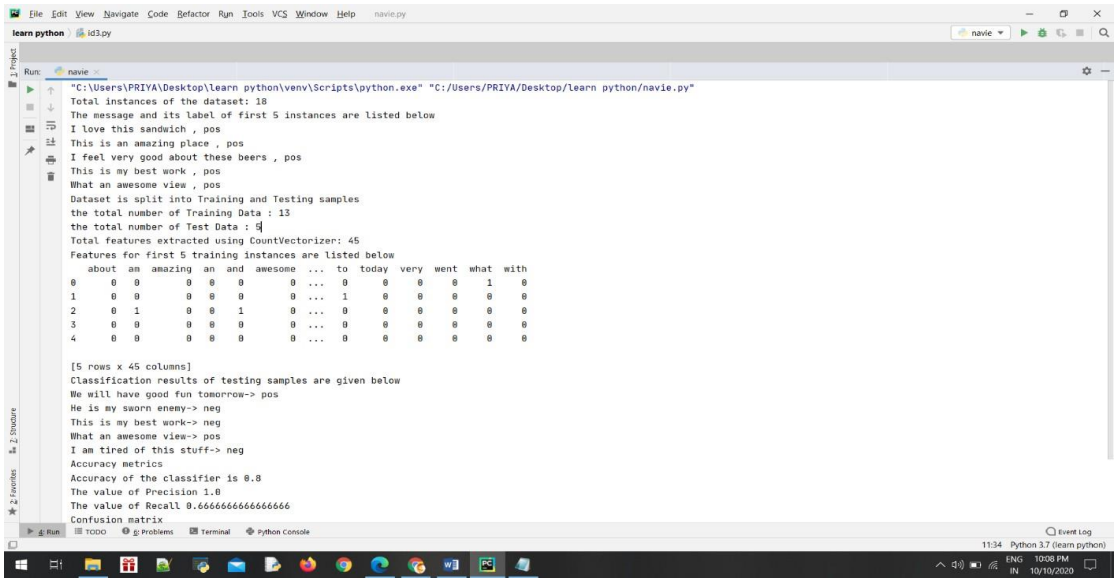
7.Assign document to class the higher probability.

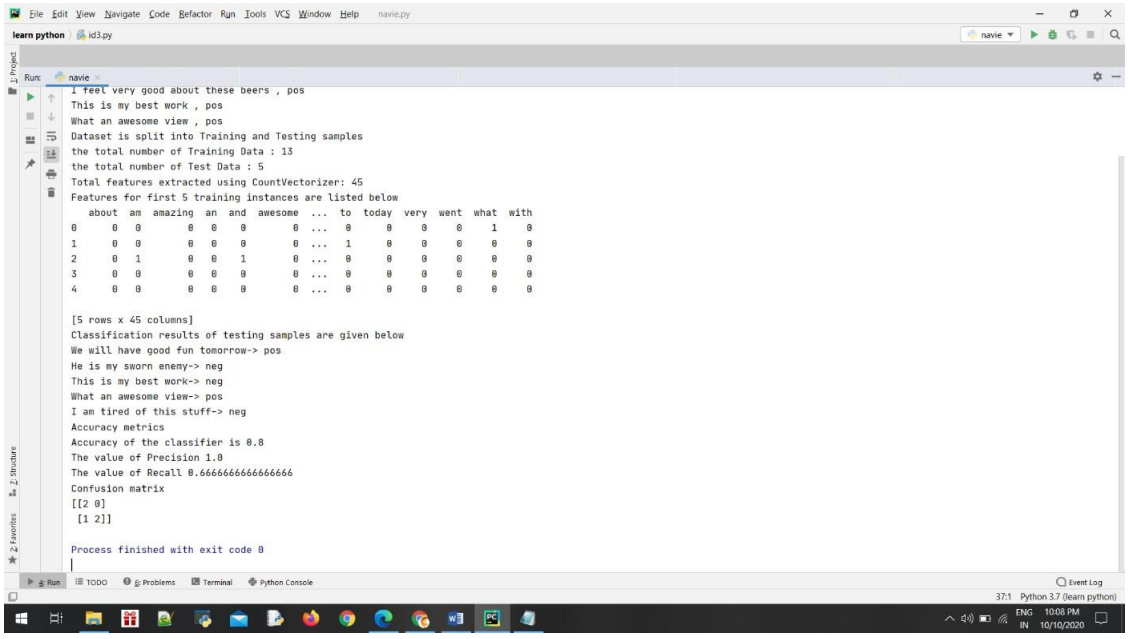
Code:

```
import pandas as pd
msg=pd.read_csv('naivetext.csv',names=['message','label'])
print('Total instances of the dataset:',msg.shape[0])
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
Y=msg.labelnum
print('The message and its label of first 5 instances are listed below')
X5, Y5 =X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
print(x, ',', y)
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X, Y)
print('Dataset is split into Training and Testing samples')
print ('the total number of Training Data :',xtrain.shape[0])
print ('the total number of Test Data :',xtest.shape[0])
from sklearn.feature_extraction.text import CountVectorizer
```

```
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('Total features extracted using CountVectorizer:',xtrain_dtm.shape[1])
print('Features for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())
print(df[0:5])
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predict(xtest_dtm)
print('Classification results of testing samples are given below')
for doc,p in zip(xtest, predicted):
pred = 'pos' if p==1 else 'neg'
print('%s-> %s'%(doc,pred))
from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('The value of Precision', metrics.precision_score(ytest,predicted))
print('The value of Recall', metrics.recall_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

Output:





Result:

Thus the naive bayes algorithm has been implemented successfully.

EXP: 8 IMPLEMENTATION OF K-NEAREST NEIGHBOUR LEARNING ALGORITHM

Aim:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Algorithm:

Input: Let m be the number of training data samples. Let p be an unknown point.

1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m

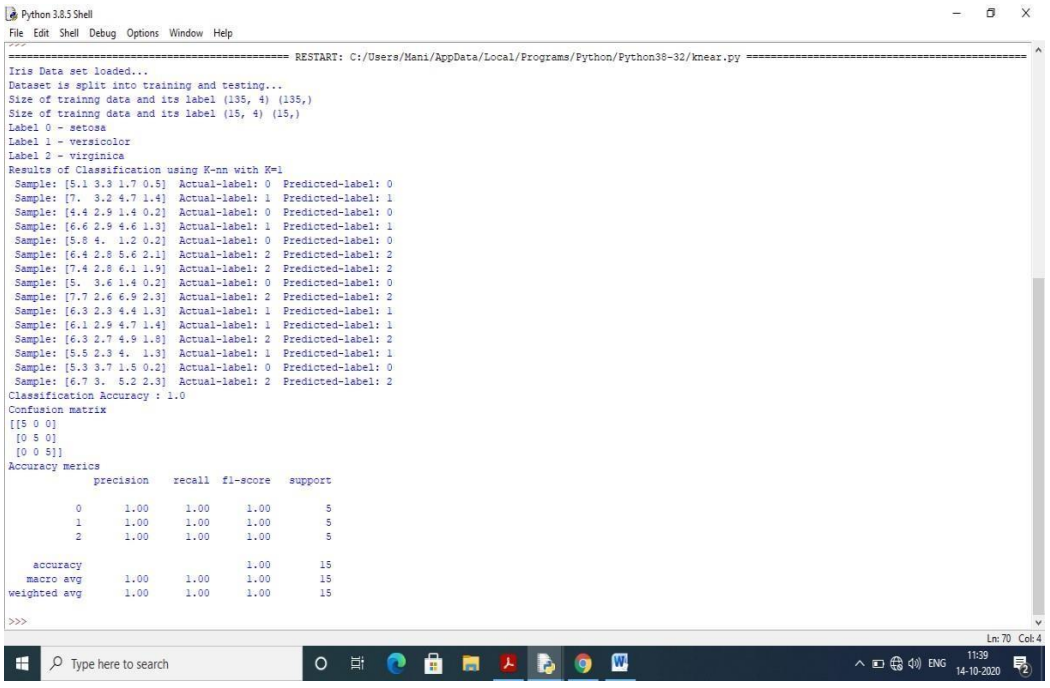
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
4. Return the majority label among S.

CODE:

```
import the required packages
from sklearn.model_selection
import train_test_split from sklearn.neighbors
import KNeighborsClassifier from sklearn
import datasets iris=datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)
for i in range(len(iris.target_names)):
print("Label", i , "-",str(iris.target_names[i]))
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
```

```
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-label:", str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));
from sklearn.metrics
import classification_report,confusion_matrix
print('Confusion matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT:



RESULT:

Thus the k-nearest neighbour learning algorithm has been implemented successfully.

EXP: 9 DESIGN AND IMPLEMENTATION OF SUPPORT VECTOR MACHINES

AIM:

To write a python program to implement the support vector machines classifier.

ALGORITHM:

1.Create a new file and save file with.py extension.

2.Import all the necessary module.

3.Import the dataset from the csv file.

4.Fit the model and train the model.

5.Split the dataset values and assign to x and y.

6.Predict the test set.

7.Print the accuracy value.

CODE:

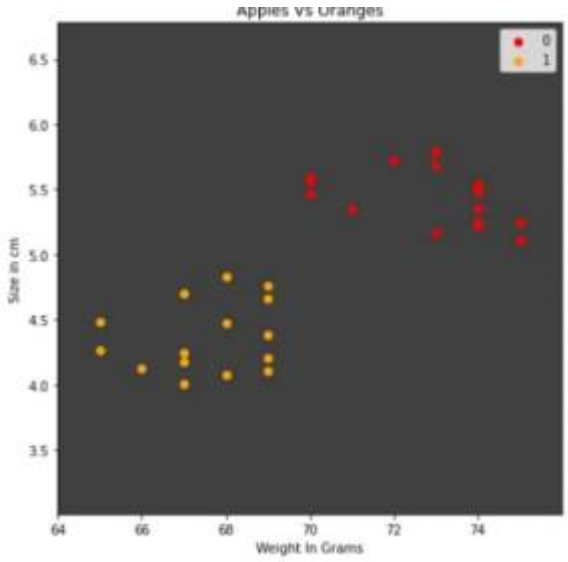
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import warnings
pd.options.mode.chained_assignment = None
data = pd.read_csv("apples_and_oranges.csv")
print(data.head())
training_set, test_set = train_test_split(data, test_size = 0.2, random_state = 1)
X_train = training_set.iloc[:,0:2].values
Y_train = training_set.iloc[:,2].values
X_test = test_set.iloc[:,0:2].values
Y_test = test_set.iloc[:,2].values
classifier = SVC(kernel='rbf', random_state = 5)
classifier.fit(X_train,Y_train)
```

```
Y_pred = classifier.predict(X_test)
test_set["Predictions"] = Y_pred
cm = confusion_matrix(Y_test,Y_pred)
accuracy = float(cm.diagonal().sum())/len(Y_test)
print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)
le = LabelEncoder()
Y_train = le.fit_transform(Y_train)
classifier = SVC(kernel='rbf', random_state = 1)
classifier.fit(X_train,Y_train)
print('Accuracy Metrics')
warnings.filterwarnings('always')
print(classification_report(Y_test,Y_pred))
print('Confusion Matrix')
print(confusion_matrix(Y_test,Y_pred))
plt.figure(figsize = (7,7))
X_set, y_set = X_train, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step =
0.01), np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('black', 'white')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'orange'))(i), label
= j)
plt.title('Apples Vs Oranges')
plt.xlabel('Weight In Grams')
plt.ylabel('Size in cm')
plt.legend()
plt.show()
```

OUTPUT:

Accuracy Metrics

	precision	recall	f1-score	support
apple	0.38	1.00	0.55	3
orange	0.00	0.00	0.00	5
accuracy			0.38	8
macro avg	0.19	0.50	0.27	8
weighted avg	0.14	0.38	0.20	8



RESULT:

Hence the design and implementation of support vector machines was executed successfully.

EXP: 10 DESIGN AND IMPLEMENTATION OF HIDDEN MARKOV MODEL

AIM:

To implement the Hidden Markov model in machine learning using python.

ALGORITHM:

1.Import all the necessary module.

2.Define the Markov edges.

3.Define a class model and initialize it.

4.Model the data using Hidden Markov Model

5.Define the variables and initialize them.

6.Define the number of states, transitions, and observations.

7.Build the transition matrix.

8.Get predictions with the Viterbi algorithm

9.Print the predictions.

CODE:

```
import os
from hmm.hmm
import DiscreteHMM
# Mapping input to variable's id
hidden_var_name = ('sunny', 'foggy', 'rainy')
observation_var_name = ('no', 'yes')
hidden_var = {
   hidden_var_name[0]: 0,
   hidden_var_name[1]: 1,
   hidden_var_name[2]: 2,
   }
observation_var = {
   observation_var_name[0]: 0,
```

```
    observation_var_name[1]: 1,
    }
target = []
obs_seq = []
with open(os.path.join(os.path.dirname(__file__), 'input.txt')) as f:
    for line in f:
        hidden, observe = line.strip().split(',')
        target.append(hidden_var[hidden])
        obs_seq.append(observation_var[observe])
# Setting model
A = (
        (0.80, 0.15, 0.05),
        (0.20, 0.50, 0.30),
        (0.20, 0.20, 0.60),
    )
B = (
        (0.9, 0.1),
        (0.7, 0.3),
        (0.2, 0.8),
    )
pi = (0.5, 0.25, 0.25)
hmm = DiscreteHMM(len(hidden_var), len(observation_var), A=A, B=B, pi=pi)
# Coculate the best explanation path by viterby algorithm
viterby_path = hmm.given(obs_seq)['viterby']
print('\n'.join([hidden_var_name[h_id] for h_id in viterby_path]))
```

OUTPUT:

```
# Setting model
B = (
        (0.8, 0.2),
        (0.5, 0.5),
        (0.1, 0.9),
    )
hmm = DiscreteHMM(len(hidden_var), len(observation_var), B=B)

# Training the model best describe the observation
hmm.train(obs_seq, verbose=1)
hmm.show_model()
print('checksum:', hmm.check_model())
```

```
itnum     1 : delta 2.248653
itnum     2 : delta 0.445165
itnum     3 : delta 0.302217
itnum     4 : delta 0.225069
itnum     5 : delta 0.177448
itnum     6 : delta 0.144390
itnum     7 : delta 0.121727
itnum     8 : delta 0.104659
itnum     9 : delta 0.092939
itnum    10 : delta 0.085674
itnum    11 : delta 0.080289
itnum    12 : delta 0.076531
itnum    13 : delta 0.074223
itnum    14 : delta 0.073234
itnum    15 : delta 0.079041
itnum    16 : delta 0.087385
itnum    17 : delta 0.096845
itnum    18 : delta 0.107395
itnum    19 : delta 0.118886
itnum    20 : delta 0.130987
```

```
B = (
        (0.8, 0.2),
        (0.5, 0.5),
        (0.1, 0.9),
    )
hmm = DiscreteHMM(len(hidden_var), len(observation_var), B=B)

# Training the model best describe the observation
hmm.train(obs_seq, verbose=1)
hmm.show_model()
print('checksum:', hmm.check_model())
```

```
itnum    93 : delta 0.017491
itnum    94 : delta 0.017280
itnum    95 : delta 0.017044
itnum    96 : delta 0.016785
itnum    97 : delta 0.016503
itnum    98 : delta 0.016349
itnum    99 : delta 0.016213
itnum   100 : delta 0.016063
----------------------A: Transition probability----------------------
[[0.7776 0.2082 0.0142]
 [0.4471 0.2763 0.2766]
 [0.3825 0.6098 0.0077]]
-----------------------B: Emission probability-----------------------
[[0.9957 0.0043]
 [0.2337 0.7663]
 [0.012  0.988 ]]
------------------pi: initital state distribution-------------------
[1. 0. 0.]
checksum: True
```

RESULT:

Thus the python program for Hidden Markov Model is implemented successfully.

EXP: 11 DESIGN ANDIMPLEMENTATION OF CLUSTERING ALGORITHM

AIM:

To write a python program to implement the clustering algorithm.

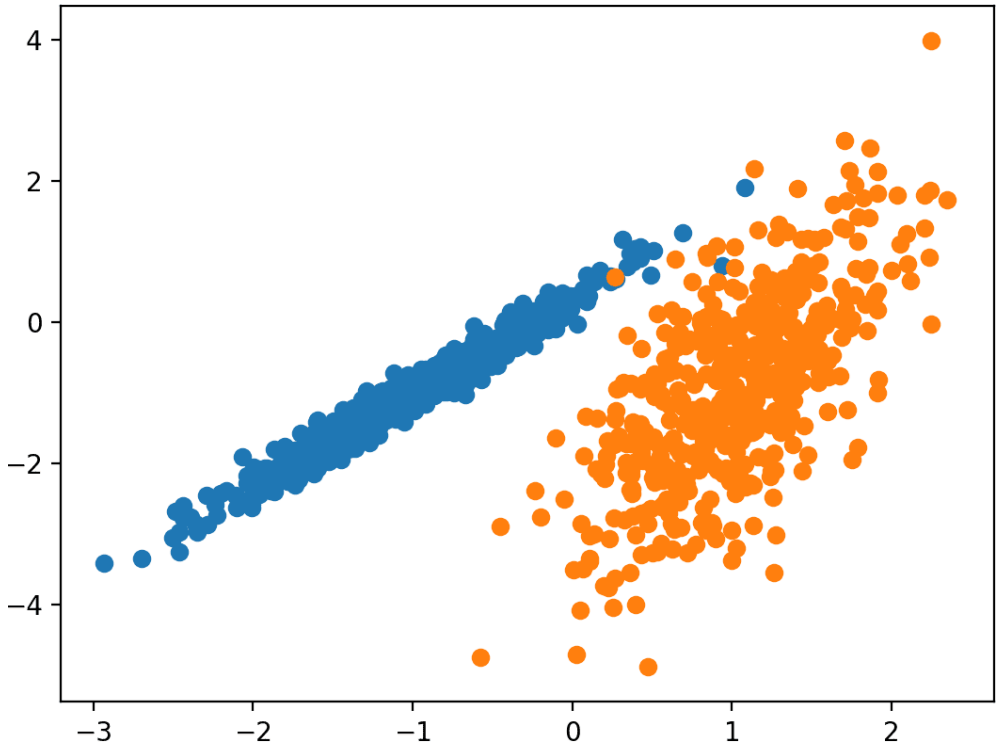
ALGORITHM:

1. Import the python libraries pandas, scalar, standards, K Means and matplotlib. Pyplot.
2. Import the dataset using pandas.
3. Standardize the data using Standard scalar(). fit_ transform() function.
4. Run the local implementation of K Means using K Means method.
5. Calculate the centroids using cluster_ centers.
6. Finally plot the clustered data using plt.subplot sandplt.scatter from plotting clusters and centroids point.

CODE:

```
From numpy import where from sklearn.datasets

import make_classification from matplotlib

import pyplot

#definedataset

X, y = make_classification(n_samples=1000,
n_features=2,n_informative=2,n_redundant=0,n_clusters_per_class=1,random_state=4)

# create scatter plot for samples from each classforclass_valueinrange(2):

        # get row indexes for samples with this classrow_ix=where(y==class_value)

        # create scatter of these samplespyplot.scatter(X[row_ix,0],X[row_ix,1])

# show the plotpyplot.show()
```

OUTPUT:



RESULT:

Hence the clustering algorithm us in python is implemented successfully.