VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

2021 Scheme Syllabus

"Machine Learning Lab"
(18AIL66)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING,

ACHARYA INSTITUTE OF TECHNOLOGY,

BANGALORE

2022-2023

COMPILED BY
MRS. Y.V.Kalyani
MR. Shivam Singh

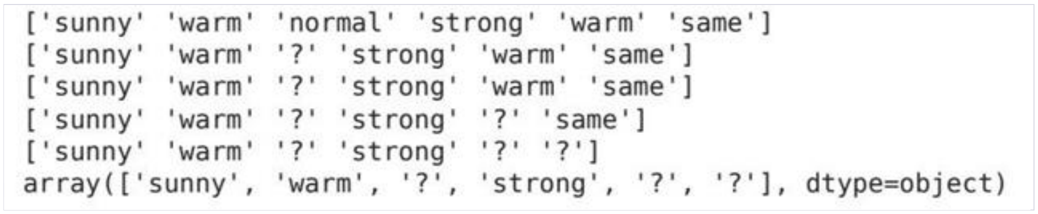
CONTENTS

PROGRAM NO.1

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file and show the output for test cases. Develop an interactive program by comparing the result by implementing LIST THEN ELIMINATE algorithm.

```
import pandas as pd
import numpy as np
def find_s(con, tar):
    for i, val in enumerate(tar):
        if val == 'Yes':
            specific_h = con[i].copy()
            break
    for i, val in enumerate(con):
        if tar[i] == "Yes":
            for j in range(len(specific_h)):
                if val[j] != specific_h[j]:
                    specific_h[j] = '?'
                else:
                    pass
    return specific_h
def list_then_eliminate(con, tar):
    general_h = ['?' for i in range(len(con[0]))]
    for i, val in enumerate(tar):
        if val == 'Yes':
            for j in range(len(con[i])):
                if general_h[j] == '?':
                    general_h[j] = con[i][j]
                elif general_h[j] != con[i][j]:
                    general_h[j] = '?'
    return general_h
data = pd.read_csv('enjoysport.csv')
print(data)
concepts=np.array(data)[:,:-1]
print(concepts)
targets=np.array(data)[:,-1]
print(targets)
h1 = find_s(concepts,targets)
print("Find_S = ",h1)
h2 = list_then_eliminate(concepts,targets)
print ("List_Then_Eliminate = ",h2)
```

OUTPUT:



```
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object)
```

PROGRAM NO. 2

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.

```
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('finds1.csv'))
concepts = np.array(data.iloc[:,0:-1])

target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    print(" steps of Candidate Elimination Algorithm",i+1)
    print("Specific_h ",i+1,"\n ")
    print(specific_h)
    print("general_h ", i+1, "\n ")
    print(general_h)

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

OUTPUT

```
initialization of specific_h and general_h
['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 8
Specific_h  8

['?' '?' '?' 'Strong' '?' '?']
general_h  8

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', 'Strong', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['?' '?' '?' 'Strong' '?' '?']
Final General_h:
[['?', '?', '?', 'Strong', '?', '?']]
```

PROGRAM NO.3

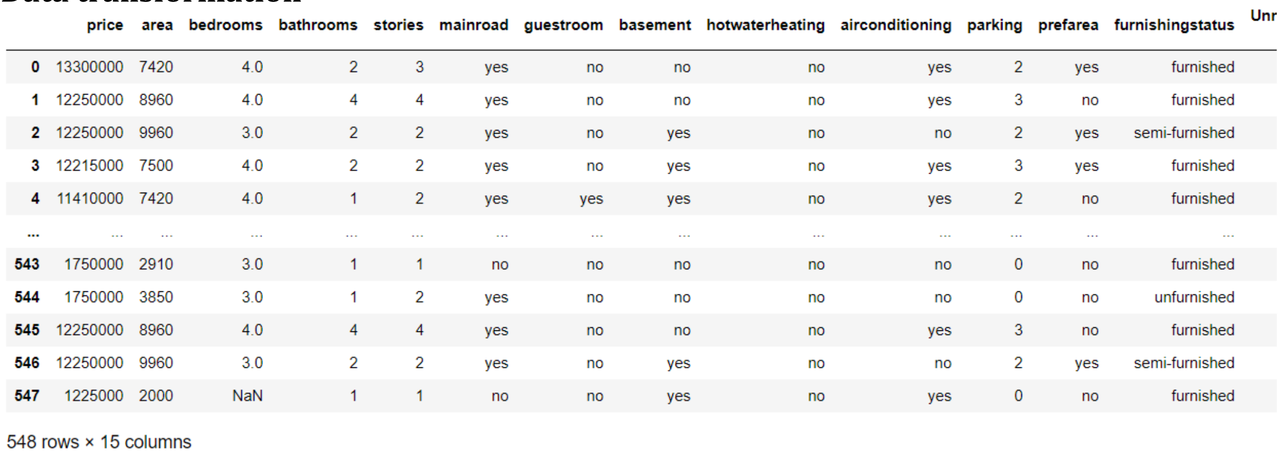
Demonstrate preprocessing(Data Cleaning, Integration and transformation)activity on suitable data.
For Ex
Identify and delete Rows that contain duplicate data by considering an appropriate dataset.
Identify and delete columns that contain a single value by considering an appropriate dataset.

```
import pandas as pd
data = pd.read_csv('Housing1.csv')
data
#Data pre processing
#data cleaning
# Check for missing values
print(data.isnull().sum())
# Remove rows with missing values
data = data.dropna()
data
#data integration
# Merge data from two datasets based on a common column
data = pd.read_csv('Housing1.csv')
data1 = pd.read_csv('Housing2.csv')
data_merged = pd.merge(data, data1, on='price')
# Concatenate two datasets vertically
data_concatenated = pd.concat([data, data1], axis=0)
data_concatenated
#Data transformation involves converting data into a suitable format or scale for analysis.
Some common techniques include:
#Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data['offer'] = scaler.fit_transform(data['offer'].values.reshape(-1, 1))
# Standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data['offer'] = scaler.fit_transform(data['offer'].values.reshape(-1, 1))
data
#single column elimination
# Create DataFrame
df = pd.DataFrame(data)
# Print original dataset
print("Original dataset:")
print(df)
print()
# Delete columns with a single value
df = df.loc[:, df.nunique() > 1]
```

```
# Print preprocessed dataset
print("Preprocessed dataset:")
print(df)
#Eliminating duplicate rows in dataset
duplicate_rows = data[data.duplicated()]
duplicate_rows
data.drop_duplicates(inplace=True)
data.reset_index(drop=True, inplace=True)
data
```

OUTPUT:

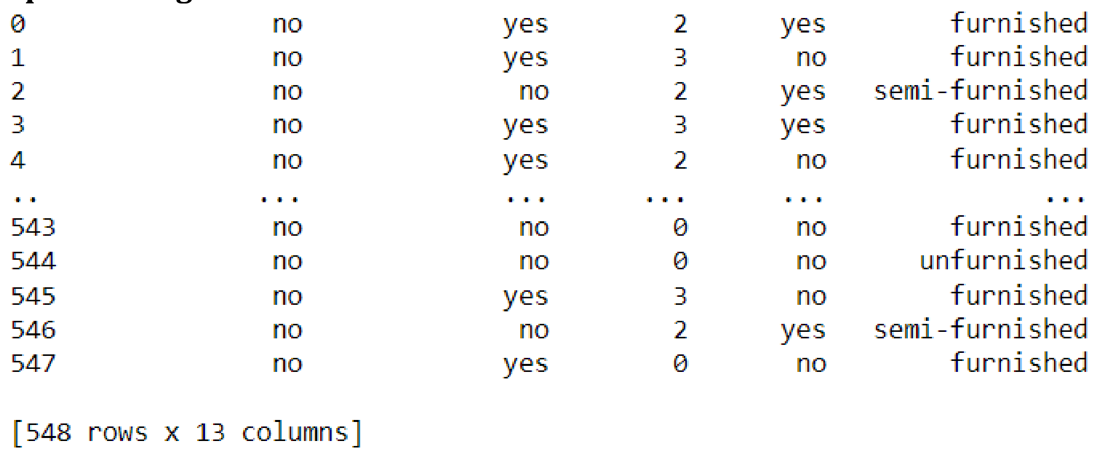
Data transformation



	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus	Unr
0	13300000	7420	4.0	2	3	yes	no	no	no	yes	2	yes	furnished	
1	12250000	8960	4.0	4	4	yes	no	no	no	yes	3	no	furnished	
2	12250000	9960	3.0	2	2	yes	no	yes	no	no	2	yes	semi-furnished	
3	12215000	7500	4.0	2	2	yes	no	yes	no	yes	3	yes	furnished	
4	11410000	7420	4.0	1	2	yes	yes	yes	no	yes	2	no	furnished	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
543	1750000	2910	3.0	1	1	no	no	no	no	no	0	no	furnished	
544	1750000	3850	3.0	1	2	yes	no	no	no	no	0	no	unfurnished	
545	12250000	8960	4.0	4	4	yes	no	no	no	yes	3	no	furnished	
546	12250000	9960	3.0	2	2	yes	no	yes	no	no	2	yes	semi-furnished	
547	1225000	2000	NaN	1	1	no	no	yes	no	yes	0	no	furnished	

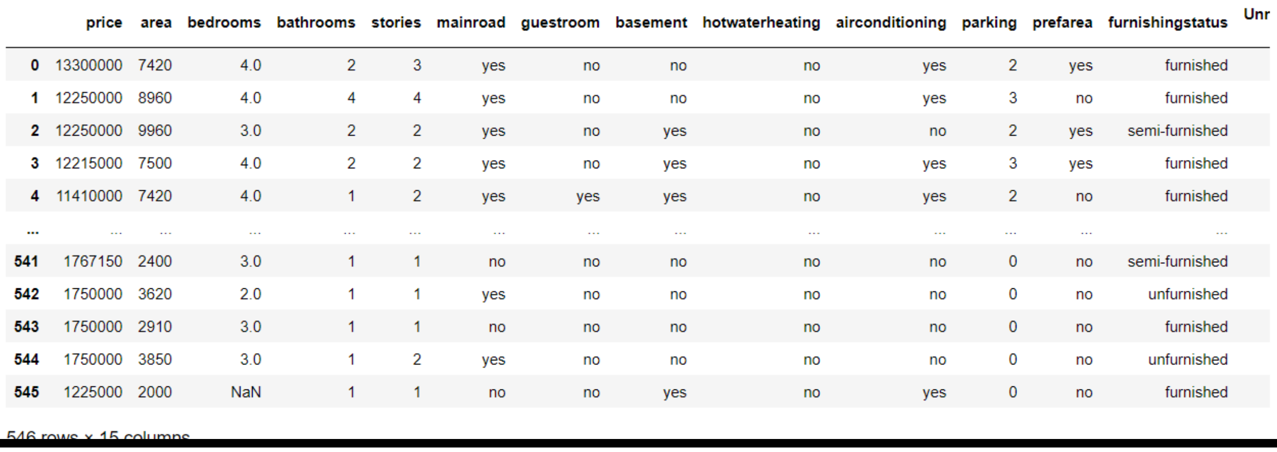
548 rows × 15 columns

Output for single column elimination



```
0                no         yes      2     yes        furnished
1                no         yes      3      no        furnished
2                no          no      2     yes   semi-furnished
3                no         yes      3     yes        furnished
4                no         yes      2      no        furnished
..              ...         ...    ...     ...              ...
543              no          no      0      no        furnished
544              no          no      0      no      unfurnished
545              no         yes      3      no        furnished
546              no          no      2     yes   semi-furnished
547              no         yes      0      no        furnished

[548 rows x 13 columns]
```

Duplicate row elimination



	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus	Unr
0	13300000	7420	4.0	2	3	yes	no	no	no	yes	2	yes	furnished	
1	12250000	8960	4.0	4	4	yes	no	no	no	yes	3	no	furnished	
2	12250000	9960	3.0	2	2	yes	no	yes	no	no	2	yes	semi-furnished	
3	12215000	7500	4.0	2	2	yes	no	yes	no	yes	3	yes	furnished	
4	11410000	7420	4.0	1	2	yes	yes	yes	no	yes	2	no	furnished	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
541	1767150	2400	3.0	1	1	no	no	no	no	no	0	no	semi-furnished	
542	1750000	3620	2.0	1	1	yes	no	no	no	no	0	no	unfurnished	
543	1750000	2910	3.0	1	1	no	no	no	no	no	0	no	furnished	
544	1750000	3850	3.0	1	2	yes	no	no	no	no	0	no	unfurnished	
545	1225000	2000	NaN	1	1	no	no	yes	no	yes	0	no	furnished	

546 rows × 15 columns

PROGRAM NO. 4

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

```
import pandas as pd
import math
# function to calculate the entropy of entire dataset
# -----------------------------------------------------------------------
def base_entropy(dataset):
    p = 0
    n = 0
    target = dataset.iloc[:, -1]
    targets = list(set(target))
    for i in target:
        if i == targets[0]:
            p = p + 1
        else:
            n = n + 1
    if p == 0 or n == 0:
        return 0
    elif p == n:
        return 1

  else:
        entropy = 0 - (
            ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
        return entropy
# -----------------------------------------------------------------------
# function to calculate the entropy of attributes
# -----------------------------------------------------------------------
def entropy(dataset, feature, attribute):
    p = 0
    n = 0
    target = dataset.iloc[:, -1]
    targets = list(set(target))
    for i, j in zip(feature, target):
        if i == attribute and j == targets[0]:
            p = p + 1
        elif i == attribute and j == targets[1]:
            n = n + 1
        if p == 0 or n == 0:
            return 0
        elif p == n:
            return 1
        else:
            entropy = 0 - (
                ((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
            return entropy
```

```
# ---------------------------------------------------------------------
# a utility function for checking purity and impurity of a child
# ---------------------------------------------------------------------
def counter(target, attribute, i):
    p = 0
    n = 0
    targets = list(set(target))
    for j, k in zip(target, attribute):
        if j == targets[0] and k == i:
            p = p + 1
        elif j == targets[1] and k == i:
            n = n + 1
    return p, n

# ---------------------------------------------------------------------
# function that calculates the information gain
# ---------------------------------------------------------------------
def Information_Gain(dataset, feature):
    Distinct = list(set(feature))
    Info_Gain = 0
    for i in Distinct:
        Info_Gain = Info_Gain + feature.count(i) / len(feature) * entropy(dataset,feature, i)
        Info_Gain = base_entropy(dataset) - Info_Gain
    return Info_Gain
# ---------------------------------------------------------------------
# function that generates the childs of selected Attribute
# ---------------------------------------------------------------------
def generate_childs(dataset, attribute_index):
    distinct = list(dataset.iloc[:, attribute_index])
    childs = dict()
    for i in distinct:
        childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute_index], i)

   return childs
# ---------------------------------------------------------------------
# function that modifies the dataset according to the impure childs
# ---------------------------------------------------------------------
def modify_data_set(dataset,index, feature, impurity):
    size = len(dataset)
    subdata = dataset[dataset[feature] == impurity]
    del (subdata[subdata.columns[index]])
    return subdata
# ---------------------------------------------------------------------
# function that return attribute with the greatest Information Gain
# ---------------------------------------------------------------------
def greatest_information_gain(dataset):
    max = -1
    attribute_index = 0
    size = len(dataset.columns) - 1
    for i in range(0, size):
        feature = list(dataset.iloc[:, i])
        i_g = Information_Gain(dataset, feature)
        if max < i_g:
            max = i_g
            attribute_index = i
    return attribute_index
# ----------------------------------------------------------------------
# function to construct the decision tree
# ----------------------------------------------------------------------
def construct_tree(dataset, tree):
    target = dataset.iloc[:, -1]
    impure_childs = []
    attribute_index = greatest_information_gain(dataset)
    childs = generate_childs(dataset, attribute_index)
    tree[dataset.columns[attribute_index]] = childs
    targets = list(set(dataset.iloc[:, -1]))
    for k, v in childs.items():
        if v[0] == 0:
            tree[k] = targets[1]
        elif v[1] == 0:
            tree[k] = targets[0]
        elif v[0] != 0 or v[1] != 0:
            impure_childs.append(k)
    for i in impure_childs:
        sub = modify_data_set(dataset,attribute_index,
        dataset.columns[attribute_index], i)
        tree = construct_tree(sub, tree)
    return tree
# -------------------------------------------------------------------------
# main function
# ----------------------------------------------------------------------
def main():
    df = pd.read_csv("playtennis.csv")
    tree = dict()
    result = construct_tree(df, tree)
    for key, value in result.items():

   print(key, " => ", value)
# ----------------------------------------------------------------------
if __name__ == "__main__":
    main()
```

OUTPUT:

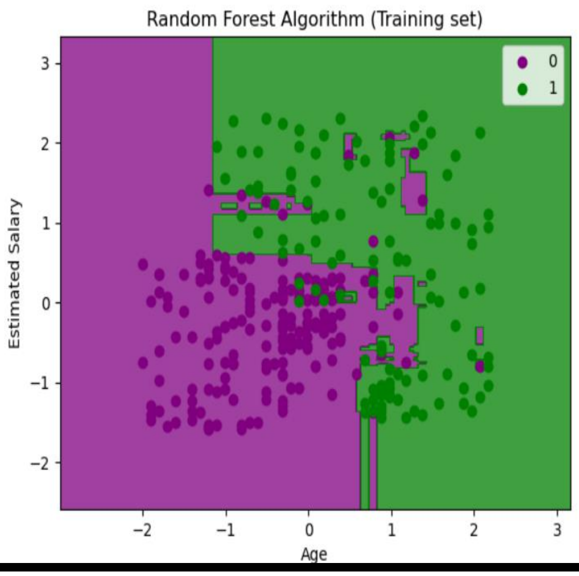
```
Outlook  =>  {'Sunny': (2, 3), 'Overcast': (4, 0), 'Rain': (3, 2)}
Overcast  =>  Yes
Temperature  =>  {'Mild': (2, 1), 'Cool': (1, 1)}
Hot  =>  No
Cool  =>  Yes
Humidity  =>  {'Normal': (1, 1)}
High  =>  No
Normal  =>  Yes
Wind  =>  {'Weak': (1, 0), 'Strong': (0, 1)}
Weak  =>  Yes
Strong  =>  No
```

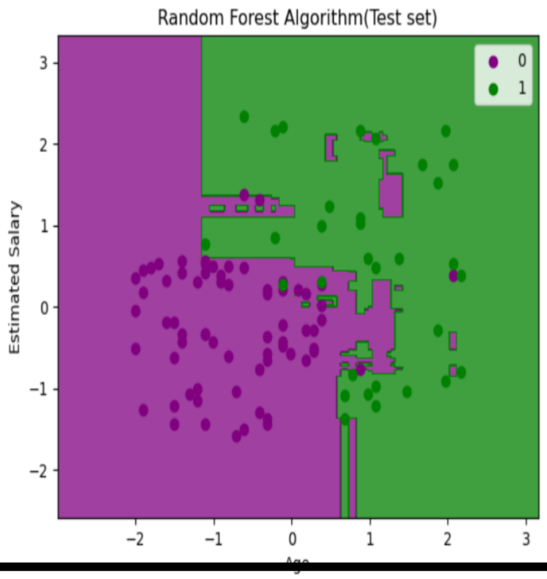
Program no.5

Demonstrate the working of the Random Forest algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('user_data.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting the dataset into training 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
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(x_test)
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT





PROGRAM NO. 6

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

```
# import necessary libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
# Load Data from CSV
data = pd.read_csv('playtennis.csv')
print("The first 5 Values of data is :\n", data.head())
# obtain train data and train output
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())
y = data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())
# convert them in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)
le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)
le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)
le_Wind = LabelEncoder()
X.Wind = le_Windy.fit_transform(X.Wind)
print("\nNow the Train output is\n", X.head())
le_PlayTennis = LabelEncoder()

y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

OUTPUT:

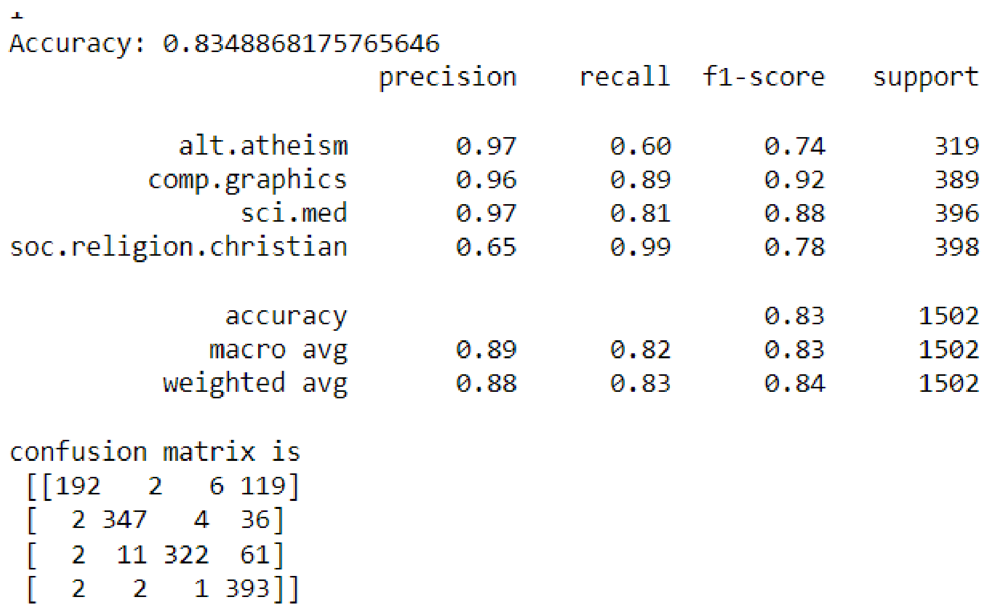
```
Accuracy is: 0.3333333333333333
```

PROGRAM NO 7

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

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)
print("\n".join(twenty_train.data[0].split("\n")))
print(twenty_train.target[0])
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_nam
es))
print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

OUTPUT:



```
Accuracy: 0.8348868175765646
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

              accuracy                           0.83      1502
             macro avg       0.89      0.82      0.83      1502
          weighted avg       0.88      0.83      0.84      1502

confusion matrix is
 [[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

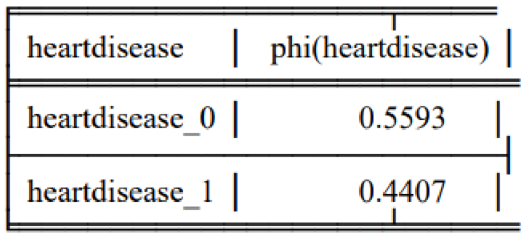
PROGRAM NO 8

Construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

```
import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
#display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())
#Model Bayesian Network
Model=BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),('tres
tbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg'),('heartdisease','thalach'),('h
eartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)
#computing the Probability of HeartDisease given Age
print('\n 1. Probability of HeartDisease given Age=30')
```

```
q=HeartDisease_infer.query(variables=['heartdisease'],evidence={'age': 37, 'sex' :0})
print(q)
```

output:



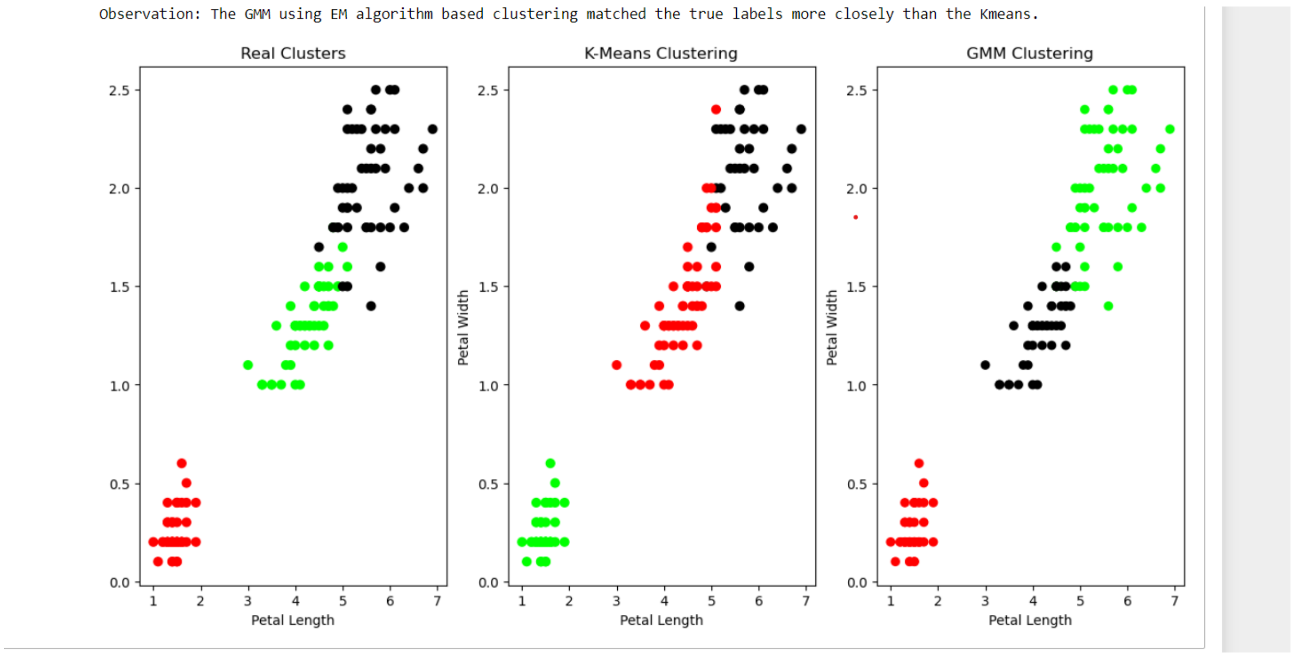
```
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease_0 |              0.5593 |
+----------------+---------------------+
| heartdisease_1 |              0.4407 |
+----------------+---------------------+
```

PROGRAM NO 9

Demonstrate the working of EM algorithm to cluster a set of data stored in .CSV file

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to
# # Visualise the clustering results
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
# Plot the Models Classifications
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.ylabel('Petal Width')

# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a # mean value 0 and standard
deviation of 1.
from sklearn.mixture import GaussianMixture
gmm= GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1, 3, 3)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm])
print('Observation: The GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.')
```

OUTPUT:

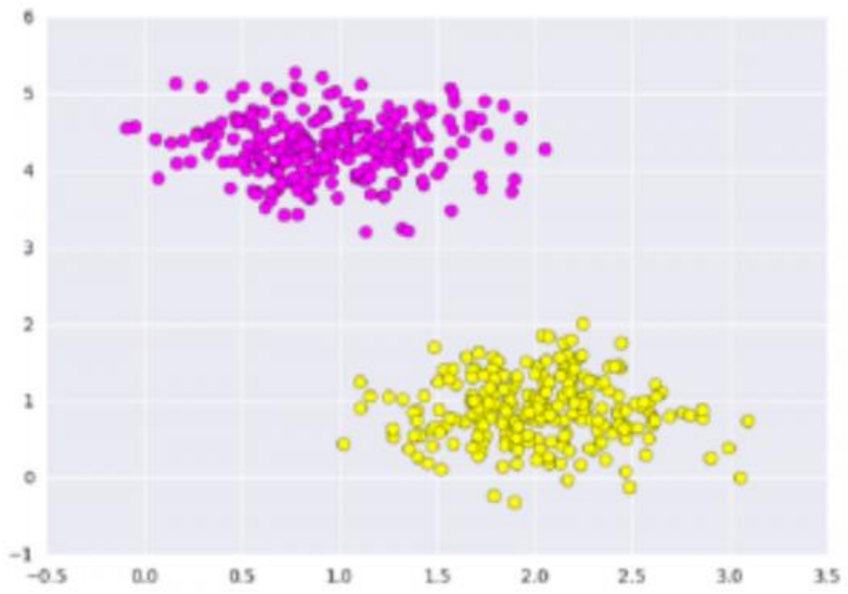


```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.
```

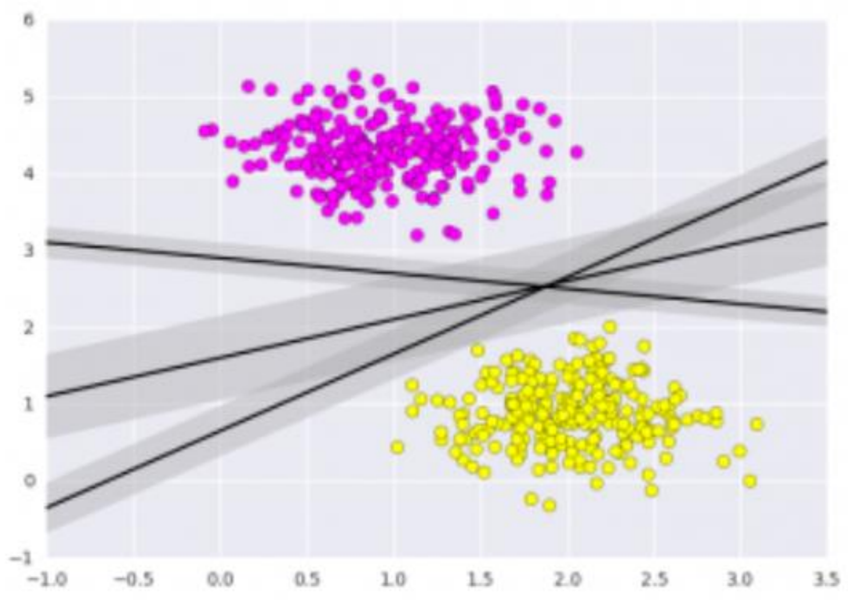
Program 10

Demonstrate the working of SVM classifier for a suitable dataset.

```
# importing scikit learn with make_blobs
from sklearn.datasets.samples_generator import make_blobs
# creating datasets X containing n_samples
# Y containing two classes
X, Y = make_blobs(n_samples=500, centers=2,
                  random_state=0, cluster_std=0.40)
import matplotlib.pyplot as plt
# plotting scatters
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```

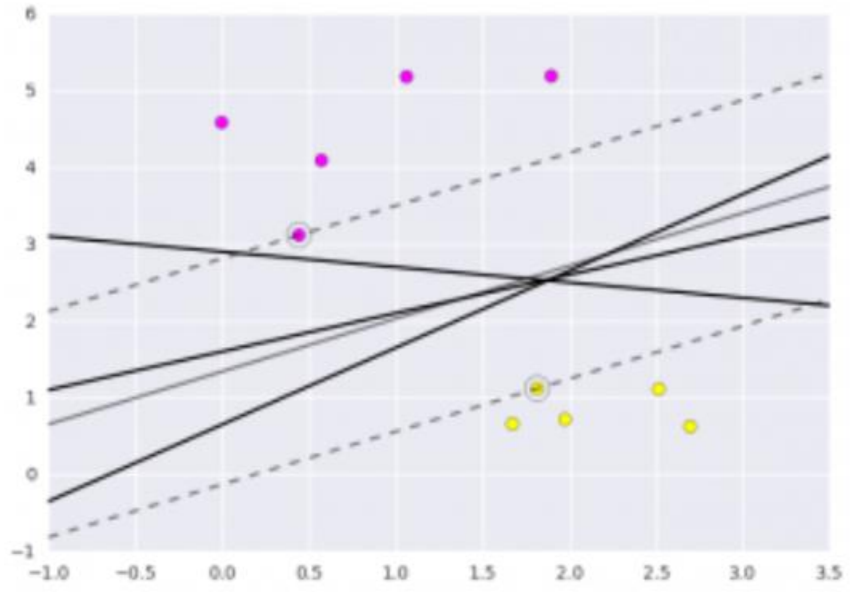


```
# creating linspace between -1 to 3.5
xfit = np.linspace(-1, 3.5)

# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
   yfit = m * xfit + b
   plt.plot(xfit, yfit, '-k')
   plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
   color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```



```
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# reading csv file and extracting class column to y.
x = pd.read_csv("C:\...\cancer.csv")
a = np.array(x)
y  = a[:,30] # classes having 0 and 1

# extracting two features
x = np.column_stack((x.malignant,x.benign))

# 569 samples and 2 features
x.shape

print (x),(y)
```

```
# import support vector classifier
# "Support Vector Classifier"
from sklearn.svm import SVC
clf = SVC(kernel='linear')

# fitting x samples and y classes
clf.fit(x, y)
clf.predict([[120, 990]])

clf.predict([[85, 550]])
```


6
5
4
3
2
1
0
−1
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5