LAB RECROD

FOR

MACHINE LEARNING LAB

Experiment 1 :- To implement decision tree learning

Dataset :- https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

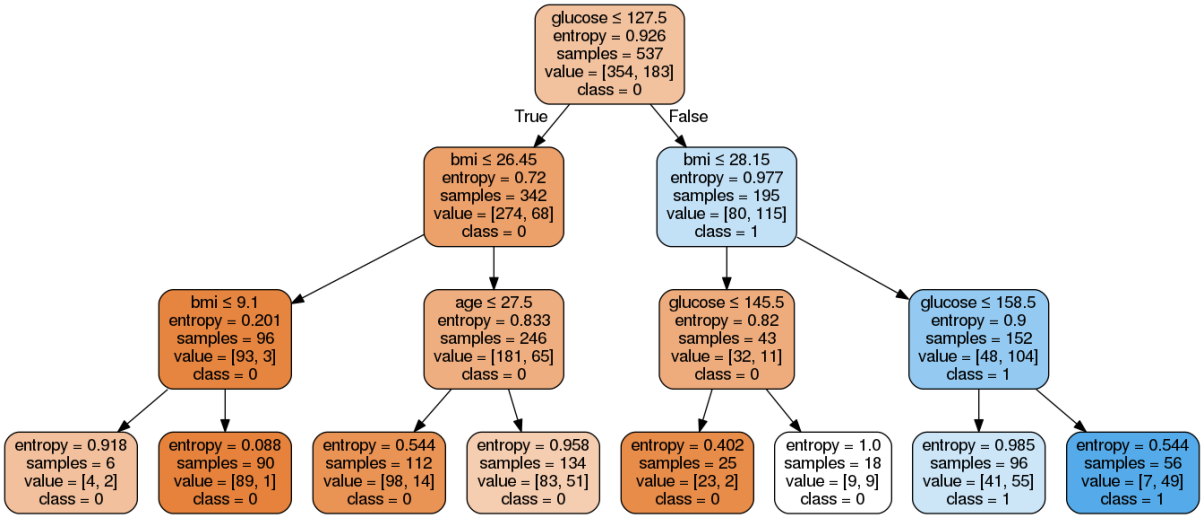
Code :-

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_names)
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# 70% training and 30% test
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
dot_data = StringIO()
```

```
export_graphviz(clf, out_file=dot_data,filled=True, rounded=True,special_characters=True,
feature_names = feature_cols,class_names=['0','1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('diabetes.png')

Image(graph.create_png())
```

Output :-

Accuracy: 0.7705627705627706



Experiment 2 :- To implement Logistic Regression

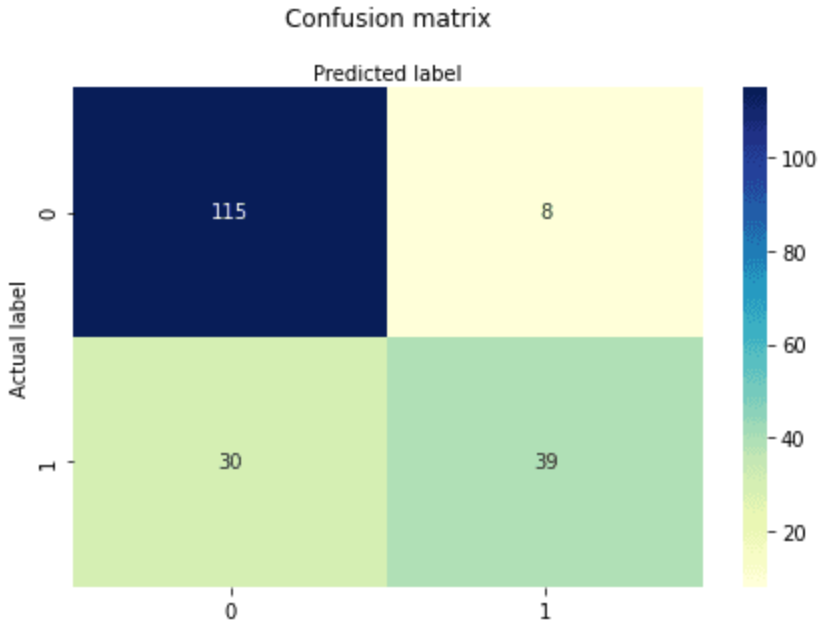
Code 1 :-

Dataset :- https://www.kaggle.com/uciml/pima-indians-diabetes-database

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_names)
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)

# instantiate the model (using the default parameters)
logreg = LogisticRegression(random_state=16)
# fit the model with data
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
```

```
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
Text(0.5,257.44,'Predicted label');
target_names = ['without diabetes', 'with diabetes']
print(classification_report(y_test, y_pred, target_names=target_names))
```

Output :-



	precision	recall	f1-score	support
without diabetes	0.79	0.93	0.86	123
with diabetes	0.83	0.57	0.67	69
accuracy			0.80	192
macro avg	0.81	0.75	0.77	192
weighted avg	0.81	0.80	0.79	192

Code 2 :-

Dataset :-

https://www.vtupulse.com/wp-content/uploads/2021/10/Data.csv

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=0, solver='warn', tol=0.0001, verbose=0,
warm_start=False)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Output :-

```
[[102, 5]
[ 3, 61]]
Accuracy: 0.9532163742690059
```

Experiment 3 :- To implement K-Nearest Neighbors Algorithm

Code:-

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
# Assigning features and label variables
# First Feature
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
# Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))
model = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
model.fit(features,label)
#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```

Output :-

```
[1]
```

KNN with Multiple Labels :-

```
#Import scikit-learn dataset library
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
#Load dataset
wine = datasets.load_wine()
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3)
# 70% training and 30% test
#Create KNN Classifier

knn = KNeighborsClassifier(n_neighbors=7)
#Train the model using the training sets
knn.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = knn.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output :-

```
Accuracy: 0.777777777778
```

Code 2:-

Dataset :-

https://drive.google.com/file/d/1dmIxI5FhAiXOQWk8Us79f_kDgzd8f0cM/view?usp=sharing

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("9-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
```

```
    else:
        print (' %-25s' % ('Wrong'))
i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

Output :-

```
  sepal-length  sepal-width  petal-length  petal-width
0        5.1         3.5          1.4          0.2
1        4.9         3.0          1.4          0.2
2        4.7         3.2          1.3          0.2
3        4.6         3.1          1.5          0.2
4        5.0         3.6          1.4          0.2


-------------------------------------------------------------------------
Original Label          Predicted Label         Correct/Wrong
-------------------------------------------------------------------------
Iris-versicolor         Iris-versicolor         Correct
Iris-virginica          Iris-versicolor         Wrong
Iris-virginica          Iris-virginica          Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-setosa             Iris-setosa             Correct
Iris-versicolor         Iris-versicolor         Correct
Iris-setosa             Iris-setosa             Correct
Iris-setosa             Iris-setosa             Correct
Iris-virginica          Iris-virginica          Correct
```

Iris-virginica Iris-versicolor Wrong

Iris-virginica Iris-virginica Correct

Iris-setosa Iris-setosa Correct

Iris-virginica Iris-virginica Correct

Iris-virginica Iris-virginica Correct

Iris-versicolor Iris-versicolor Correct

Confusion Matrix:

[[4 0 0]

[0 4 0]

[0 2 5]]

Classification Report:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	4
Iris-versicolor	0.67	1.00	0.80	4
Iris-virginica	1.00	0.71	0.83	7
avg / total	0.91	0.87	0.87	15

Accuracy of the classifer is 0.87

Experiment 4 :-Implement classification using Naïve-Bayes

Code :-

Dataset :-

https://drive.google.com/file/d/124kwlLWIw0XK4dZaEisPYVvqDKSIDB85/view?usp=sharing

```
import csv
import random
import math
def loadcsv(filename):
	lines = csv.reader(open(filename, "r"));
	dataset = list(lines)
	for i in range(len(dataset)):
		dataset[i] = [float(x) for x in dataset[i]]
	return dataset
def splitdataset(dataset, splitratio):
	trainsize = int(len(dataset) * splitratio);
	trainset = []
	copy = list(dataset);
	while len(trainset) <trainsize:
		index = random.randrange(len(copy));
		trainset.append(copy.pop(index))
	return [trainset, copy]
def separatebyclass(dataset):
	separated = {} #dictionary of classes 1 and 0
	#creates a dictionary of classes 1 and 0 where the values are
	#the instances belonging to each class
	for i in range(len(dataset)):
		vector = dataset[i]
		if (vector[-1] not in separated):
			separated[vector[-1]] = []
		separated[vector[-1]].append(vector)
```

```
        return separated
def mean(numbers):
        return sum(numbers)/float(len(numbers))
def stdev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)
def summarize(dataset): #creates a dictionary of classes
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
        del summaries[-1] #excluding labels +ve or -ve
        return summaries
def summarizebyclass(dataset):
        separated = separatebyclass(dataset);
        summaries = {}
        for classvalue, instances in separated.items():
                summaries[classvalue] = summarize(instances) #used to cal to mean and std
        return summaries
def calculateprobability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
def calculateclassprobabilities(summaries, inputvector):
        probabilities = {} # probabilities contains the all prob of all class of test data
        for classvalue, classsummaries in summaries.items():
                probabilities[classvalue] = 1
                for i in range(len(classsummaries)):
                        mean, stdev = classsummaries[i]
                        x = inputvector[i] #testvector's first attribute
                        probabilities[classvalue] *= calculateprobability(x, mean, stdev);
        return probabilities
def predict(summaries, inputvector): #training and test data is passed
        probabilities = calculateclassprobabilities(summaries, inputvector)
```

```
        bestLabel, bestProb = None, -1
        for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
                if bestLabel is None or probability >bestProb:
                        bestProb = probability
                        bestLabel = classvalue
        return bestLabel
def getpredictions(summaries, testset):
        predictions = []
        for i in range(len(testset)):
                result = predict(summaries, testset[i])
                predictions.append(result)
        return predictions
def getaccuracy(testset, predictions):
        correct = 0
        for i in range(len(testset)):
                if testset[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testset))) * 100.0
def main():
        filename = 'naivedata.csv'
        splitratio = 0.67
        dataset = loadcsv(filename);
        trainingset, testset = splitdataset(dataset, splitratio)
        print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))
        # prepare model
        summaries = summarizebyclass(trainingset);
        predictions = getpredictions(summaries, testset)
        #find the predictions of test data with the training data
        accuracy = getaccuracy(testset, predictions)
```

```
        print('Accuracy of the classifier is : {0}%'.format(accuracy))
main()
```

Output :-

Split 768 rows into train=514 and test=254

Rows Accuracy of the classifier is : 71.65354330708661%

Code 2 :-Demonstrate Bayesian network using pgmpy

Dataset :-

https://drive.google.com/file/d/17vwRLAY8uR-6vWusM5prn08it-BEGp-f/view?usp=sharing

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heart disease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output :-

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Experiment 5 :-To implement classification using SVM

Code :-#SVM with scikit learn

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
#Load dataset
cancer = datasets.load_breast_cancer()
# Split dataset into training set and test set
X_train,X_test,y_train,y_test=train_test_split(cancer.data,cancer.target,test_size=0.3,random_state
=109)
 # 70% training and 30% test
#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel
#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))
# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

Output :-

```
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

Code 2 :-

Dataset :-

https://www.vtupulse.com/wp-content/uploads/2021/10/Data.csv

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Output :-

```
[[102, 5]
[ 5, 59]]
Accuracy: 0.9415204678362573
```

Experiment 6 :-To implement principle component analysis for Dimensionality Reduction

Code :-

Output :-

Experiment 7 :- To implement Apriori principle for Association analysis

Code :-

Dataset :-

https://archive.ics.uci.edu/ml/datasets/Online+Retail

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data1 = pnd.read_excel('Online_Retail.xlsx')
# here, we will strip the extra spaces in the description
data1['Description'] = data1['Description'].str.strip()
# Now, drop the rows which does not have any invoice number
data1.dropna(axis = 0, subset = ['InvoiceNo'], inplace = True)
data1['InvoiceNo'] = data1['InvoiceNo'].astype('str')
# Now, we will drop all transactions which were done on credit
data1 = data1[~data1['InvoiceNo'].str.contains('C')]
# Transactions done in France
basket1_France =(data1[data1['Country']=="France"].groupby(['InvoiceNo',
'Description'])['Quantity'] .sum().unstack().reset_index().fillna(0).set_index('InvoiceNo'))
# Transactions done in the United Kingdom
basket1_UK=(data1[data1['Country']=="UnitedKingdom"].groupby(['InvoiceNo',
Description'])['Quantity'].sum().unstack().reset_index().fillna(0).set_index('InvoiceNo'))
# Transactions done in Portugal
basket1_Por = (data1[data1['Country'] == "Portugal"] .groupby(['InvoiceNo',
'Description'])['Quantity'] .sum().unstack().reset_index().fillna(0).set_index('InvoiceNo'))
basket1_Sweden=(data1[data1['Country']=="Sweden"].groupby(['InvoiceNo',
'Description'])['Quantity'].sum().unstack().reset_index().fillna(0).set_index('InvoiceNo'))
def hot_encode1(P):
  if(P<= 0):
    return 0
  if(P>= 1):
    return 1
```

```
# Here, we will encode the datasets
basket1_encoded = basket1_France.applymap(hot_encode1)
basket1_France = basket1_encoded
basket1_encoded = basket1_UK.applymap(hot_encode1)
basket1_UK = basket1_encoded
basket1_encoded = basket1_Por.applymap(hot_encode1)
basket1_Por = basket1_encoded
basket1_encoded = basket1_Sweden.applymap(hot_encode1)
basket1_Sweden = basket1_encoded
# Build the model  for FRANCE
frq_items1 = AP(basket1_France, min_support = 0.05, use_colnames = True)
# Collect the inferred rules in a dataframe
rules1 = AR(frq_items1, metric = "lift", min_threshold = 1)
rules1 = rules1.sort_values(['confidence', 'lift'], ascending = [False, False])
print(rules1.head()) #Inferred rules for France
#Similarly we can get the rules for UK,Portugal,Sweden
```

Output :-

antecedents \

45 (JUMBO BAG WOODLAND ANIMALS)

260 (PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ...

272 (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...

302 (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...

301 (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...

consequents antecedent support consequent support \

45 (POSTAGE) 0.076531 0.765306

260 (POSTAGE) 0.051020 0.765306

272 (POSTAGE) 0.053571 0.765306

302 (SET/6 RED SPOTTY PAPER PLATES) 0.102041s 0.127551

301 (SET/6 RED SPOTTY PAPER CUPS) 0.102041 0.137755

support confidence lift leverage conviction

45 0.076531 1.000 1.306667 0.017961 inf

260 0.051020 1.000 1.306667 0.011974 inf

272 0.053571 1.000 1.306667 0.012573 inf

302 0.099490 0.975 7.644000 0.086474 34.897959

301 0.099490 0.975 7.077778 0.085433 34.489796

Experiment 8 :-Implement K-means Clustering to Find Natural Patterns in Data

Code :-

Dataset :-

https://drive.google.com/file/d/1dmIxI5FhAiXOQWk8Us79f_kDgzd8f0cM/view?usp=sharing

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']
dataset = pd.read_csv("8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])
# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])
print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))
# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
```

```
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output :-

```
The accuracy score of K-Mean: 0.24
The Confusion matrixof K-Mean:
[[ 0 50 0]
[48 0 2]
[14 0 36]]
The accuracy score of EM: 0.36666666666666664
The Confusion matrix of EM:
[[50 0 0]
[ 0 5 45]
[ 0 50 0]]
```

Experiment 9 :-To implement Hierarchical clustering

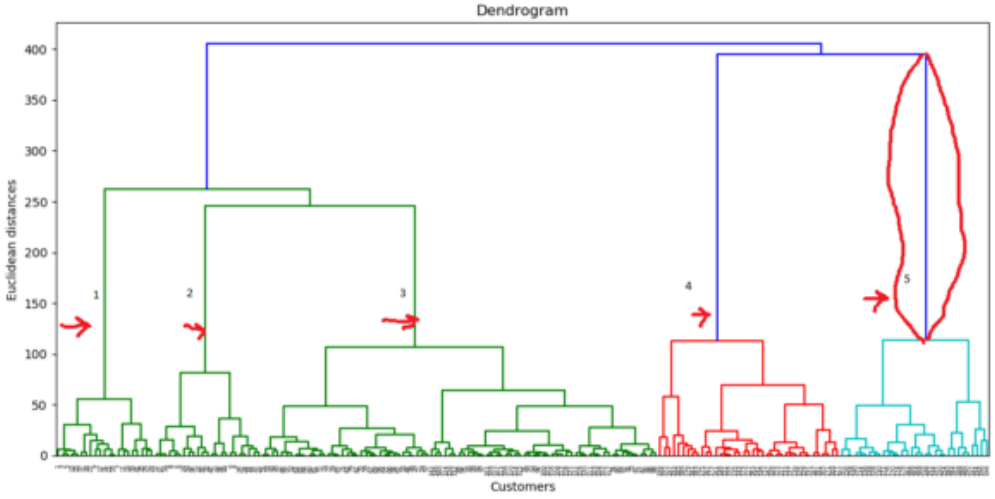
Code :-

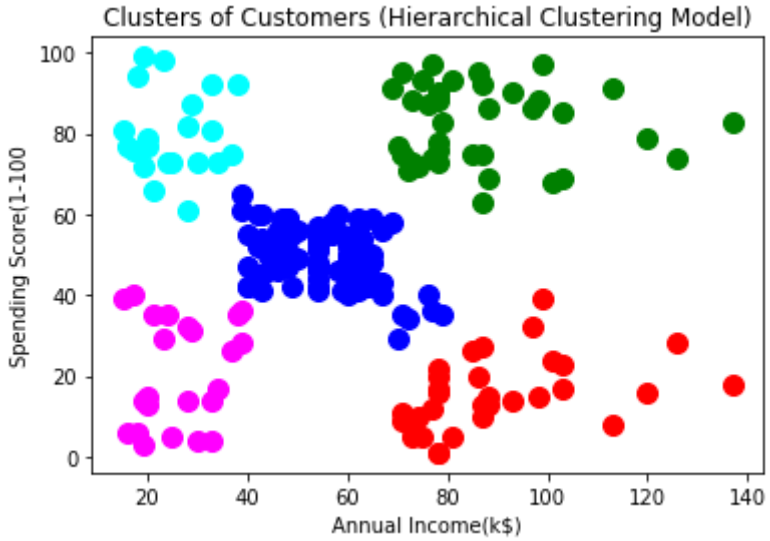
Dataset :-

https://github.com/Hrd2D/Hierarchical-Clustering-Algorithm-in-Machine-Learning-using-Python/blob/main/Mall_Customers.csv

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
#2 Importing the Mall_Customers dataset by pandas
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3,4]].values
dendrogram = sch.dendrogram(sch.linkage(X, method  = "ward"))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage ='ward')
y_hc=hc.fit_predict(X)
plt.scatter(X[y_hc==0, 0], X[y_hc==0, 1], s=100, c='red', label ='Cluster 1')
plt.scatter(X[y_hc==1, 0], X[y_hc==1, 1], s=100, c='blue', label ='Cluster 2')
plt.scatter(X[y_hc==2, 0], X[y_hc==2, 1], s=100, c='green', label ='Cluster 3')
plt.scatter(X[y_hc==3, 0], X[y_hc==3, 1], s=100, c='cyan', label ='Cluster 4')
plt.scatter(X[y_hc==4, 0], X[y_hc==4, 1], s=100, c='magenta', label ='Cluster 5')
plt.title('Clusters of Customers (Hierarchical Clustering Model)')
plt.xlabel('Annual Income(k$)')
plt.ylabel('Spending Score(1-100')
plt.show()
```

Output :-





Cluster1(Red), Cluster2 (Blue), Cluster3(Green), Cluster4(Cyan), Cluster5 (Magenta)

Experiment 10 :-To implement Random Forests (Bagging).

Code :-

Dataset :-

https://www.vtupulse.com/wp-content/uploads/2021/10/Data.csv

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False, random_state=0, verbose=0,
warm_start=False)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy of Random Forest Classifier:',accuracy_score(y_test, y_pred))
```

Output :-

Confusion Matrix of Random Forest Classifier

[[102, 5]

[6, 58]]

Accuracy of Random Forest Classifier: 0.935672514619883

Experiment 11 :-To implement Adaboost (Boosting)

Code :-

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import train_test_split
from sklearn.datasets import make_hastie_10_2
import matplotlib.pyplot as plt

""" HELPER FUNCTION: GET ERROR RATE ========================================="""
def get_error_rate(pred, Y):
    return sum(pred != Y) / float(len(Y))

""" HELPER FUNCTION: PRINT ERROR RATE ======================================="""
def print_error_rate(err):
    print 'Error rate: Training: %.4f - Test: %.4f' % err

""" HELPER FUNCTION: GENERIC CLASSIFIER ====================================="""
def generic_clf(Y_train, X_train, Y_test, X_test, clf):
clf.fit(X_train,Y_train)
pred_train = clf.predict(X_train)
pred_test = clf.predict(X_test)
    return get_error_rate(pred_train, Y_train), \
get_error_rate(pred_test, Y_test)

""" ADABOOST IMPLEMENTATION ================================================="""
def adaboost_clf(Y_train, X_train, Y_test, X_test, M, clf):
n_train, n_test = len(X_train), len(X_test)
    # Initialize weights
    w = np.ones(n_train) / n_train
```

```
pred_train, pred_test = [np.zeros(n_train), np.zeros(n_test)]


   for i in range(M):
      # Fit a classifier with the specific weights
clf.fit(X_train, Y_train, sample_weight = w)
pred_train_i = clf.predict(X_train)
pred_test_i = clf.predict(X_test)
      # Indicator function
      miss = [int(x) for x in (pred_train_i != Y_train)]
      # Equivalent with 1/-1 to update weights
      miss2 = [x if x==1 else -1 for x in miss]
      # Error
err_m = np.dot(w,miss) / sum(w)
      # Alpha
alpha_m = 0.5 * np.log( (1 - err_m) / float(err_m))
      # New weights
      w = np.multiply(w, np.exp([float(x) * alpha_m for x in miss2]))
      # Add to prediction
pred_train = [sum(x) for x in zip(pred_train,
                                  [x * alpha_m for x in pred_train_i])]
pred_test = [sum(x) for x in zip(pred_test,
                                 [x * alpha_m for x in pred_test_i])]


pred_train, pred_test = np.sign(pred_train), np.sign(pred_test)
   # Return error rate in train and test set
   return get_error_rate(pred_train, Y_train), \
get_error_rate(pred_test, Y_test)


""" PLOT FUNCTION ============================================================="""
def plot_error_rate(er_train, er_test):
df_error = pd.DataFrame([er_train, er_test]).T
```

```
df_error.columns = ['Training', 'Test']
    plot1 = df_error.plot(linewidth = 3, figsize = (8,6),
          color = ['lightblue', 'darkblue'], grid = True)
    plot1.set_xlabel('Number of iterations', fontsize = 12)
    plot1.set_xticklabels(range(0,450,50))
    plot1.set_ylabel('Error rate', fontsize = 12)
    plot1.set_title('Error rate vs number of iterations', fontsize = 16)
plt.axhline(y=er_test[0], linewidth=1, color = 'red', ls = 'dashed')


""" MAIN SCRIPT =============================================================="""
if __name__ == '__main__':


    # Read data
    x, y = make_hastie_10_2()
df = pd.DataFrame(x)
df['Y'] = y


    # Split into training and test set
    train, test = train_test_split(df, test_size = 0.2)
X_train, Y_train = train.ix[:,:-1], train.ix[:,-1]
X_test, Y_test = test.ix[:,:-1], test.ix[:,-1]


    # Fit a simple decision tree first
clf_tree = DecisionTreeClassifier(max_depth = 1, random_state = 1)
er_tree = generic_clf(Y_train, X_train, Y_test, X_test, clf_tree)


    # Fit Adaboost classifier using a decision tree as base estimator
    # Test with different number of iterations
er_train, er_test = [er_tree[0]], [er_tree[1]]
x_range = range(10, 410, 10)
    for i in x_range:
```

```
er_i = adaboost_clf(Y_train, X_train, Y_test, X_test, i, clf_tree)
er_train.append(er_i[0])
er_test.append(er_i[1])


   # Compare error rate vs number of iterations
plot_error_rate(er_train, er_test)
```

Output :-

Experiment 12 :-Evaluating ML algorithm with balanced and unbalanced datasets Comparison of MachineLearning algorithms. Develop a mini project with any one of the Machine LearningModels.

Code :-

Output :-