Lab-3 [Naive Bayes]

Algorithm :-

1) Input the data set from the user in the form of a CSV / excel file.

2) Convert the table into a frequency table.

3) Calculate the prior probabilities for various classes. Then using the Naive bayes formula calc. the posterior probability.

$$P(C/x) = \frac{P(x/C) \cdot P(C)}{P(x)}$$

where; $P(C)$ = Class - prior prob.
$P(C/x)$ = Posterior prob.

4) Now, the class with highest posterior probability to be pitched as outcome of prediction.

Code :-

Assigning features and label variables

Weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast' 'Overcast', 'Rainy']

Temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild' 'Hot', 'Mild']

Play = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

Import LabelEncoder from sklearn import preprocessing

Creating labelEncoder le = preprocessing.LabelEncoder()

Converting string labels into numbers

```
weather_encoded = le.fit_transform(weather)
print(weather_encoded)

temp_encoded = le.fit_transform(temp)
label = le.fit_transform(play)
print("temp:", temp_encoded)
print("play:", label)

# Combining weather and temp into single list of tuples

features = list(zip(weather_encoded, temp_encoded))
print(features)

# Import Gaussian Naive Bayes Model
from sklearn.naive_bayes import GaussianNB

# Create a Gaussian Classifier
model = GaussianNB()

# Train the model set using the training sets
model.fit(features, label)

# Predict Output
predicted = model.predict([0,2])

# 0: Overcast, 2: Mild
print("predicted value:", predicted)
```

Lab-2

[Decision Tree]

Algorithm :-

1) It begins with original set of data as root node.

2) On each iteration of algo, it iterates through the very unused attribute of the dataset and calculates Entropy and Info. gain of the attributes.

3) It then selects the attribute which has the smallest entropy or largest information gain.

4) The set is then split by selected attribute to produce a subset of data.

5) The algo. recursively make new decision trees using the subsets of data created in step-4. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as leaf node.

Code:-

```
import pandas as pd
from sklearn import tree
from sklearn.model_selection import
    train_test_split
from sklearn.externals import joblib
from sklearn.Metrics import accuracy_score

data = pd.read_excel('dataset.xlsx')
data.head()

# converting categorical values into binary values
data_bin = pd.get_dummies(data)

# input data
X = data_bin.drop columns = ["Play_no",
                             "Play_yes"]

# output data
y = data["Play"]
y2 = pd.get_dummies(y)

# splitting data into train and test

X_train, X_test, y_train, y_test = train_test_split
                    (X, y2, test_size = 0.25,
                     random_state = 50)
```

Creating model and fitting it with the training data

```
Model_dt = tree.DecisionTreeClassifier
                (criterion="entropy")

Model_dt.fit(x_train, y_train)
```

predicting and accuracy

```
pred = Model_dt.predict(x_test)

score = accuracy_score(y_test, pred)
score
```

Lab - 06 [KNN]

PAGE NO
DATE :

Algorithm :-

1) First we need to Create or Import a dataset

2) Now we need to choose the value of K

3) Now from each point in the test data, calc the distance b/w test data and each row of training data with the help of Euclidean distance.

4) Now based on the distance value, sort them

5) Since the distance values are sorted, it will pick K values from sorted array.

6) Now, it will assign a class to the test point based on most frequent class among these rows.

Code :-

```
Import pandas as pd
import math
```

creating dataset

```
dataset = pd.dataFrame()

dataset['Data Mining'] = [4, 6, 7, 5, 8]
dataset['OS'] = [3, 7, 8, 5, 8]

dataset['Result'] = ['fail', 'pass', 'pass',
                     'fail', 'pass']
dataset
```

value of K

```
K = int(input())
```

inputed query

```
dm_marks = int(input())
os_marks = int(input())

no_rows = len(dataset.index)

list_Eucdist = list()
list_target = list()

for i range(no_rows):

    P1 = dataset.iloc[i, 0]
    P2 = dataset.iloc[i, 1]
```

PAGE NO
DATE

```
distance = Math.sqrt(Math.Pow(abs(dm-marks-P1),2),
                     + Math.Pow(abs(os-marks-P2),2))

list_target.append(dataset.idoc[i,2])
list-EucDist.append(str(distance))

print("Euclidian distance for X", i+1,";=",
                    distance)

list1 = list()
list1 = sorted(zip(list_EucDist, list_target))

# variables for counting no. of pass & fail
pass_no = 0
fail_no = 0

for i in range(K):
    if list1[i][:] = "pass":
        pass_no += 1

    else:
        fail_no += 1

# final Result

result = "pass" if pass_no > fail_no
else "fail"
print("According to dataset and input the
      result is=", result)
```