|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment No:** | 05 |  |  |  |  |  | TE AI&DS | |
| Date of Performance: | |  |  |  |  |  | Roll No: 9696 | |
| **Aim**: To Implement Naïve Bayesian classification algorithm to build classification model and predict class for unseen data. | | | | | | | | |
| **Related CO5:** Implement Classification, Clustering and Association mining techniques to extract knowledge | | | | | | | | |
| **Objective:**  To learn how classification is used for Prediction. | | | | | | | | |
| **Rubrics for assessment of Experiment:** | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **Sr. No** | **Parameters** | **Exceed** | |  | **Meet** | | **Below** | |
|  |  | **Expectations(EE)** | | | **Expectations** | | **Expectations** | |
|  |  |  |  |  | **(ME)** | | **(BE)** | |
| 1 | Timeline (2) | Early or on time | | | One session late | | More than one | |
|  |  | (2) |  |  | (1) |  | session late (0) | |
| 2 | Preparedness (2) | Knows the basic | | | Managed to | | Not aware of the | |
|  |  | theory related to | | | explain the | | theory to the | |
|  |  | the experiment | | | theory related to | | point. (1) | |
|  |  | very well. (2) | |  | the experiment. | |  |  |
|  |  |  |  |  | (1) |  |  |  |
| 3 | Effort (3) | Done expt on their | | | Done expt with | | Just managed. | |
|  |  | own. (3) | |  | help from other. | | (1) |  |
|  |  |  |  |  | (2) |  |  |  |
| 4 | Documentation(2) | Lab experiment is | | | Documented in | | Experiments not | |
|  |  | documented in | | | proper format | | written in | |
|  |  | proper format and | | | but some | | proper format | |
|  |  | maintained neatly. | | | formatting | | (0.5) |  |
|  |  | (2) |  |  | guidelines are | |  |  |
|  |  |  |  |  | missed. (1) | |  |  |
| 5 | Result (1) | Specific | |  | Partially | | Not specific at | |
|  |  | conclusion.(1) | | | specific | | all. (0) | |
|  |  |  |  |  | conclusion. | |  |  |
|  |  |  |  |  | (0.5) |  |  |  |
| **Assessment Marks:** | |  |  |  |  |  |  |  |
|  |  |  |  |  | |  | |  |
| Timeline(2) | Preparedness(2) | Effort(3) |  | Documentation(2) | | Result(1) | | Total(10) |
|  |  |  |  |  |  |  |  |  |

**Theory:**

**Naive Bayes classifiers** are a family of “probabilistic classifiers” based on [Bayes’ theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with strong independence between the features. They are among the simplest Bayesian network models and are capable of achieving high accuracy levels.

Bayes theorem states mathematically as:  
P(A|B) = ( P(B|A) \* P(A) )/ P(B)  
where A and B are events and P(B) != 0.  
P(A|B) is a conditional probability: the probability of event A occurring given that B is true.  
P(B|A) is also a conditional probability: the probability of event B occurring given that A is true.  
P(A) and P(B) are the probabilities of observing A and B respectively without any given conditions.  
A and B must be different events.

**Bayes Theorem** 

* Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event
* conditional probability can be found this way
* Assume we have a Hypothesis(*H*) and evidence(*E*),   
  According to Bayes theorem, the relationship between the probability of Hypothesis before getting the evidence represented as *P(H)* and the probability of the hypothesis after getting the evidence represented as *P(H|E)* is:

*P(H|E) = P(E|H)\*P(H)/P(E)*

* **Prior probability** = *P(H)* is the probability before getting the evidence   
  **Posterior probability** = *P(H|E)* is the probability after getting evidence
* In general,

*P(class|data) = (P(data|class) \* P(class)) / P(data)*

**Bayes Theorem Example**   
Assume we have to find the probability of the randomly picked card to be king given that it is a face card.   
There are *4* Kings in a Deck of Cards which implies that *P(King) = 4/52*   
as all the Kings are face Cards so *P(Face|King) = 1*   
there are *3* Face Cards in a Suit of *13 cards* and there are *4 Suits* in total so *P(Face) = 12/52*   
Therefore,

*P(King|face) = P(face|king)\*P(king)/P(face) = 1/3*

We can frame classification as a conditional classification problem with Bayes Theorem as follows:

* P(yi | x1, x2, …, xn) = P(x1, x2, …, xn | yi) \* P(yi) / P(x1, x2, …, xn)

The prior *P(yi)* is easy to estimate from a dataset, but the conditional probability of the observation based on the class *P(x1, x2, …, xn | yi)* is not feasible unless the number of examples is extraordinarily large, e.g. large enough to effectively estimate the probability distribution for all different possible combinations of values.

Example

**#Weather Dataset**

**Outlook Temp Humidity Windy Play**

Rainy Hot High f no

Rainy Hot High t no

Overcast Hot High f yes

Sunny Mild High f yes

Sunny Cool Normal f yes

Sunny Cool Normal t no

Overcast Cool Normal t yes

Rainy Mild High f no

Rainy Cool Normal f yes

Sunny Mild Normal f yes

Rainy Mild Normal t yes

Overcast Mild High t yes

Overcast Hot Normal f yes

Sunny Mild High t no

Calculate Prior Probability of Classes P(y)

**#Frequency tableP**

**(Play=Yes)** = 9/14 = 0.64  
**P(Play=No)** = 5/14 = 0.36

## Calculate the Likelihood Table for all features

**#Likelihood Table#Outlook**

**Play Overcast Rainy Sunny**

Yes 4/9 2/9 3/9

No 0/5 3/5 2/5

\_\_\_ \_\_\_ \_\_\_

4/14 5/14 5/14

**#Temp**

**Play Cool Mild Hot**

Yes 3/9 4/9 2/9

No 1/5 2/5 2/5

\_\_\_ \_\_\_ \_\_\_

4/14 6/14 4/14

**#Humidity**

**Play High Normal**

Yes 3/9 6/9

No 4/5 1/5

\_\_\_ \_\_\_

7/14 7/14

**#Windy**

**Play f t**

Yes 6/9 3/9

No 2/5 3/5

\_\_\_ \_\_\_

8/14 6/14

## Now, Calculate Posterior Probability for each class using the Naive Bayesian equation. The Class with maximum probability is the outcome of the prediction.

**Query:**Whether Players will play or not when the weather conditions are [Outlook=Rainy, Temp=Mild, Humidity=Normal, Windy=t]?

**Calculation of Posterior Probability:**

**Since Conditional independence of two random variables, A and B gave C holds just in case  
P(A, B | C) = P(A | C) \* P(B | C)**

**P(y=Yes|x)** = **P(Yes|Rainy,Mild,Normal,t)**

P(Rainy,Mild,Normal,t|Yes) \* P(Yes)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

P(Rainy,Mild,Normal,t)

P(Rainy|Yes)\*P(Mild|Yes)\*P(Normal|Yes)\*P(t|Yes)\*P(Yes)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

P(Rainy)\*P(Mild)\*P(Normal)\*P(t)

(2/9) \* (4/9) \* (6/9) \* (3/9) \* (9/14)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(5/14) \* (6/14) \* (7/14) \* (6/14)

= **0.43**

**P(y=No|x)** = **P(No|Rainy,Mild,Normal,t)**

P(Rainy,Mild,Normal,t|No) \* P(No)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

P(Rainy,Mild,Normal,t)

P(Rainy|No)\*P(Mild|No)\*P(Normal|No)\*P(t|No)\*P(No)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

P(Rainy)\*P(Mild)\*P(Normal)\*P(t)

(3/5) \* (2/5) \* (1/5) \* (3/5) \* (5/14)

= \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(5/14) \* (6/14) \* (7/14) \* (6/14)

= **0.31**

Now, **P(Play=Yes|Rainy,Mild,Normal,t**) has the highest Posterior probability.

Algorithm/steps:

1. Load the dataset and convert it into list.
2. Separating the data as per class(0,1)-mp[0], mp[1]
3. define the test input: test = [2,1,0,1]
4. find the prior probability of each class
5. Find the conditional probability of test for each class-[yes,no]

probYes = 1 // probNO = 1

count = 0

total = 0

**//find the length of test**

**for** i **in** range(len(test)):

count = 0

total = 0

**for** row **in** mp[1]: //mp[0] for NO

**if**(test[i] == row[i]):

count += 1

total += 1

probYes \*= count/total //probNO\*= count/total

6) Display the posterior probability of each class and find maximization

**Code with output:**

Code:-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import math

def accuracy\_score(y\_true, y\_pred):

""" score = (y\_true - y\_pred) / len(y\_true) """

return round(float(sum(y\_pred == y\_true))/float(len(y\_true)) \* 100 ,2)

def pre\_processing(df):

""" partioning data into features and target """

X = df.drop([df.columns[-1]], axis = 1)

y = df[df.columns[-1]]

return X, y

class NaiveBayes:

"""

Bayes Theorem:

Likelihood \* Class prior probability

Posterior Probability = -------------------------------------

Predictor prior probability

P(x|c) \* p(c)

P(c|x) = ------------------

P(x)

"""

def \_\_init\_\_(self):

"""

Attributes:

likelihoods: Likelihood of each feature per class

class\_priors: Prior probabilities of classes

pred\_priors: Prior probabilities of features

features: All features of dataset

"""

self.features = list

self.likelihoods = {}

self.class\_priors = {}

self.pred\_priors = {}

self.X\_train = np.array

self.y\_train = np.array

self.train\_size = int

self.num\_feats = int

def fit(self, X, y):

self.features = list(X.columns)

self.X\_train = X

self.y\_train = y

self.train\_size = X.shape[0]

self.num\_feats = X.shape[1]

for feature in self.features:

self.likelihoods[feature] = {}

self.pred\_priors[feature] = {}

for feat\_val in np.unique(self.X\_train[feature]):

self.pred\_priors[feature].update({feat\_val: 0})

for outcome in np.unique(self.y\_train):

self.likelihoods[feature].update({feat\_val+'\_'+outcome:0})

self.class\_priors.update({outcome: 0})

self.\_calc\_class\_prior()

self.\_calc\_likelihoods()

self.\_calc\_predictor\_prior()

def \_calc\_class\_prior(self):

""" P(c) - Prior Class Probability """

for outcome in np.unique(self.y\_train):

outcome\_count = sum(self.y\_train == outcome)

self.class\_priors[outcome] = outcome\_count / self.train\_size

def \_calc\_likelihoods(self):

""" P(x|c) - Likelihood """

for feature in self.features:

for outcome in np.unique(self.y\_train):

outcome\_count = sum(self.y\_train == outcome)

feat\_likelihood = self.X\_train[feature][self.y\_train[self.y\_train == outcome].index.values.tolist()].value\_counts().to\_dict()

for feat\_val, count in feat\_likelihood.items():

self.likelihoods[feature][feat\_val + '\_' + outcome] = count/outcome\_count

def \_calc\_predictor\_prior(self):

""" P(x) - Evidence """

for feature in self.features:

feat\_vals = self.X\_train[feature].value\_counts().to\_dict()

for feat\_val, count in feat\_vals.items():

self.pred\_priors[feature][feat\_val] = count/self.train\_size

def predict(self, X):

""" Calculates Posterior probability P(c|x) """

results = []

X = np.array(X)

for query in X:

probs\_outcome = {}

for outcome in np.unique(self.y\_train):

prior = self.class\_priors[outcome]

likelihood = 1

evidence = 1

for feat, feat\_val in zip(self.features, query):

likelihood \*= self.likelihoods[feat][feat\_val + '\_' + outcome]

evidence \*= self.pred\_priors[feat][feat\_val]

posterior = (likelihood \* prior) / (evidence)

probs\_outcome[outcome] = posterior

result = max(probs\_outcome, key = lambda x: probs\_outcome[x])

results.append(result)

return np.array(results)

if \_\_name\_\_ == "\_\_main\_\_":

#Weather Dataset

print("\nWeather Dataset:")

df = pd.read\_table("Weather\_dataset.txt")

#print(df)

#Split fearures and target

X,y = pre\_processing(df)

nb\_clf = NaiveBayes()

nb\_clf.fit(X, y)

print("Train Accuracy: {}".format(accuracy\_score(y, nb\_clf.predict(X))))

#Query:

query = np.array([['Rainy','Mild', 'Normal', 't']])

print("Query 1:- {} ---> {}".format(query, nb\_clf.predict(query)))

# Output:-

# Weather Dataset:

# Train Accuracy: 92.4

# Query 1:- [['Rainy' 'Mild' 'Normal' 't']] ---> ['Overcast Cool Normal t yes']

Part 2:

*import* pandas *as* pd

*from* sklearn.naive\_bayes *import* CategoricalNB

*from* sklearn.preprocessing *import* LabelEncoder

*from* sklearn.metrics *import* accuracy\_score

*# Load Weather Dataset*

df = pd.read\_csv("Weather\_dataset.txt")

*# Encode the target variable (class labels)*

le = LabelEncoder()

y = le.fit\_transform(df['Play'])

*# Encode the categorical features*

X = df.drop('Play', *axis*=1).apply(le.transform)

*# Create and train the Naive Bayes classifier*

nb\_clf = CategoricalNB()

nb\_clf.fit(X, y)

*# Predict and calculate accuracy on the training data*

y\_pred = nb\_clf.predict(X)

accuracy = accuracy\_score(y, y\_pred)

print("Train Accuracy: {*:.2f*}%".format(accuracy \* 100))

*# Query:*

query = pd.DataFrame({'Outlook': ['Rainy'], 'Temperature': ['Mild'], 'Humidity': ['Normal'], 'Windy': ['False']})

*# Use the same LabelEncoder for query data*

query\_encoded = query.apply(*lambda* *col*: le.transform(col))

prediction = nb\_clf.predict(query\_encoded)

print("Query 1:", le.inverse\_transform(prediction))

# Output:-

# Weather Dataset:

# Train Accuracy: 92.4

# Query 1:- [['Rainy' 'Mild' 'Normal' 't']] ---> ['Overcast Cool Normal t yes']

References:

[Naive Bayes Classification Program in Python from Scratch - japp.io](https://japp.io/machine-learning/naive-bayes-classification-program-in-python-from-scratch/)

[Naïve Bayes Algorithm -Implementation from scratch in Python. | by ranga\_vamsi | Medium](https://medium.com/@rangavamsi5/na%C3%AFve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9)

[ML | Naive Bayes Scratch Implementation using Python - GeeksforGeeks](https://www.geeksforgeeks.org/ml-naive-bayes-scratch-implementation-using-python/)

[How to Develop a Naive Bayes Classifier from Scratch in Python (machinelearningmastery.com)](https://machinelearningmastery.com/classification-as-conditional-probability-and-the-naive-bayes-algorithm/)

**Conclusion:**

**I am able to understand the navie bayes theorem and am able to implement the code**