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 Expectations(EE)	Meet Expectations (ME)	Below Expectations (BE)
1	Timeline (2)	Early or on time (2)	One session late (1)	More than one session late (0)
2	Preparedness (2)	Knows the basic theory related to the experiment very well. (2)	Managed to explain the theory related to the experiment. (1)	Not aware of the theory to the point. (1)
3	Effort (3)	Done expt on their own. (3)	Done expt with help from other. (2)	Just managed. (1)
4	Documentation(2)	Lab experiment is documented in proper format and maintained neatly. (2)	Documented in proper format but some formatting guidelines are missed. (1)	Experiments not written in proper format (0.5)
5	Result (1)	Specific conclusion.(1)	Partially specific conclusion. (0.5)	Not specific at all. (0)

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 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:

```python
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)
```

```python
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
Naïve Bayes Algorithm -Implementation from scratch in Python. | by ranga_vamsi | Medium

ML | Naive Bayes Scratch Implementation using Python - GeeksforGeeks
How to Develop a Naive Bayes Classifier from Scratch in Python (machinelearningmastery.com)

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