EXPERIMENT NO. 4

AIM: Implementation of Bayesian Classification Algorithm..

Software used: Java / C/ Python

Theory

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).

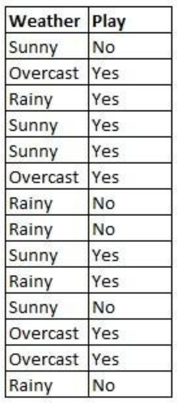
$$\underbrace{P(c\mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\mid \mathrm{X}) = P(x_1\mid c)\times P(x_2\mid c)\times \cdots \times P(x_n\mid c)\times P(c)$$

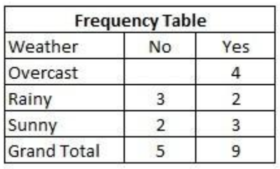
Above, a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition.

Step 1: Convert the data set into a frequency table

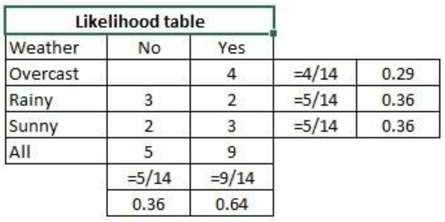
Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.



Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No



Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9



Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.
- **Step 3**: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.
- **Problem:** Players will play if weather is sunny. Is this statement correct?

We can solve it using above discussed method of posterior probability.

- P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)
- Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
- Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Advantages:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Disadvantages:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a

prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Applications of Naive Bayes Algorithms

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

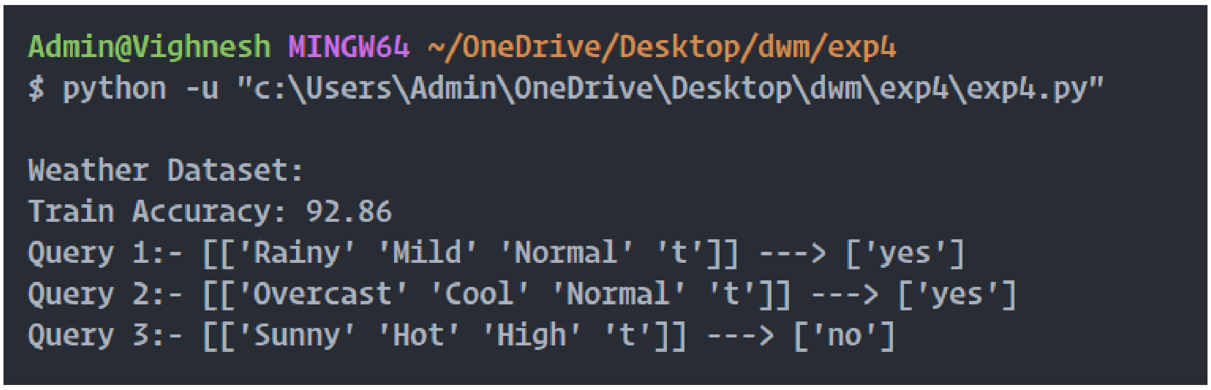
PROGRAM:

```
import numpy as np
import pandas as pd
from matplotlib import 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:
	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__":
        print("\nWeather Dataset:")
        df = pd.read_csv('./play.csv')
        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 1:
        query = np.array([['Rainy','Mild', 'Normal', 't']])
        print("Query 1:- {} ---> {}".format(query, nb_clf.predict(query)))
        #Query 2:
        query = np.array([['Overcast','Cool', 'Normal', 't']])
        print("Query 2:- {} ---> {}".format(query, nb_clf.predict(query)))
        #Query 3:
        query = np.array([['Sunny','Hot', 'High', 't']])
        print("Query 3:- {} ---> {}".format(query, nb_clf.predict(query)))
```

INPUT & OUTPUT:



```
Admin@Vighnesh MINGW64 ~/OneDrive/Desktop/dwm/exp4
$ python -u "c:\Users\Admin\OneDrive\Desktop\dwm\exp4\exp4.py"

Weather Dataset:
Train Accuracy: 92.86
Query 1:- [['Rainy' 'Mild' 'Normal' 't']] ---> ['yes']
Query 2:- [['Overcast' 'Cool' 'Normal' 't']] ---> ['yes']
Query 3:- [['Sunny' 'Hot' 'High' 't']] ---> ['no']
```

CONCLUSION:

Thus we performed Bayesian Classification

SIGN AND REMARK

R1 (3 M)	R2 (3 M)	R3 (3 M)	R4 (3 M)	R5 (3 M)	Total	Sign

DATE