**Batch:B2**

**Roll Number: 16010420061 Experiment No:3**

**Name:Sargundeep Sachdeo**

**Title of the Experiment:Implementation of Naïve Bayesian algorithm for classification**

**Program:**

**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**

**def train\_test\_split(x, y, test\_size = 0.25, random\_state = None):**

**""" partioning the data into train and test sets """**

**x\_test = x.sample(frac = test\_size, random\_state = random\_state)**

**y\_test = y[x\_test.index]**

**x\_train = x.drop(x\_test.index)**

**y\_train = y.drop(y\_test.index)**

**return x\_train, x\_test, y\_train, y\_test**

**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()**

***# print(self.likelihoods)***

***# print(self.class\_priors)***

***# print(self.pred\_priors)***

**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("Data\weather.txt")**

***#print(df)***

***#Split fearures and target***

**X,y = pre\_processing(df)**

***#Split data into Training and Testing Sets***

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.1, random\_state = 0)**

***#print(X\_train, y\_train)***

**nb\_clf = NaiveBayes()**

**nb\_clf.fit(X\_train, y\_train)**

***#print(X\_train, y\_train)***

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

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

***#Query 1:***

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

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

***#Query 2:***

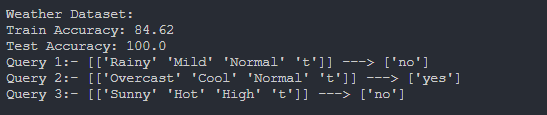
**query = np.array([['Overcast','Cool', 'Normal', 't']])**

**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)))**

**Output:  
**

**Post Lab Question- Answers (If Any):**

**Q.1. What are advantages and disadvantages of Bayesian Classification?**

**Ans:** Advantages

* This algorithm works quickly and can save a lot of time.
* Naive Bayes is suitable for solving multi-class prediction problems.
* If its assumption of the independence of features holds true, it can perform better than other models and requires much less training data.
* Naive Bayes is better suited for categorical input variables than numerical variables.

### Disadvantages

### Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.

### This algorithm faces the ‘zero-frequency problem’ where it assigns zero probability to a categorical variable whose category in the test data set wasn’t available in the training dataset. It would be best if you used a smoothing technique to overcome this issue.

### Its estimations can be wrong in some cases, so you shouldn’t take its probability outputs very seriously.

**Q.2. Comment on Laplacian correction.**

**Ans:** Laplace correction is a smoothing technique that helps tackle the problem of zero probability in the Naïve Bayes machine learning algorithm. Using higher alpha values will push the likelihood towards a value of 0.5, i.e., the probability of a word equal to 0.5 for both the positive and negative reviews.

**CO:** Comprehend basics of ML

**Conclusion:** In this experiment, I successfully understood and implemented Data preprocessing techniques.