Mahavir Education Trust's

SHAH & ANCHOR KUTCHHI ENGINEERING COLLEGE

Chembur, Mumbai - 400 088

UG Program in Cyber Security

<table>
<tr><th colspan="6">Experiment Number: 4</th></tr>
<tr><td colspan="2">Date of Performance:</td><td colspan="4"></td></tr>
<tr><td colspan="2">Date of Submission:</td><td colspan="4"></td></tr>
<tr><td>Program Execution/ formation/ correction/ ethical practices (07)</td><td>Documentation (02)</td><td>Timely Submission (03)</td><td>Viva Answer to sample questions (03)</td><td>Experiment Total (15)</td><td>Sign</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Experiment 4

Aim: Implementation of Bayesian algorithm

Lab outcomes: CSL 503.2: Implement data mining algorithms like classification.

Problem Statement: Implement the Naive Bayes algorithm.

Theory:

Bayes theorem: Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

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.

Types

Gaussian Naive Bayes

This type of Naive Bayes is used when variables are continuous in nature. It assumes that all the variables have a normal distribution. So if you have some variables which do not have this property, you might want to transform them to the features having distribution normal.

Multinomial Naive Bayes

This is used when the features represent the frequency.

Suppose you have a text document and you extract all the unique words and create multiple features where each feature represents the count of the word in the document. In such a case, we have a frequency as a feature. In such a scenario, we use multinomial Naive Bayes. It ignores the non-occurrence of the features. So, if you have frequency 0 then the probability of occurrence of that feature will be 0 hence multinomial naive Bayes ignores that feature. It is known to work well with text classification problems.

Bernoulli Naive Bayes

This is used when features are binary. So, instead of using the frequency of the word, if you have discrete features in 1s and 0s that represent the presence or absence of a feature. In that case, the features will be binary and we will use Bernoulli Naive Bayes.

Algorithm:

- Step 1: Separate by Class.
- Step 2: Summarize Dataset.
- Step 3: Summarize Data by Class.
- Step 4: Gaussian Probability Density Function.
- Step 5: Class Probabilities.

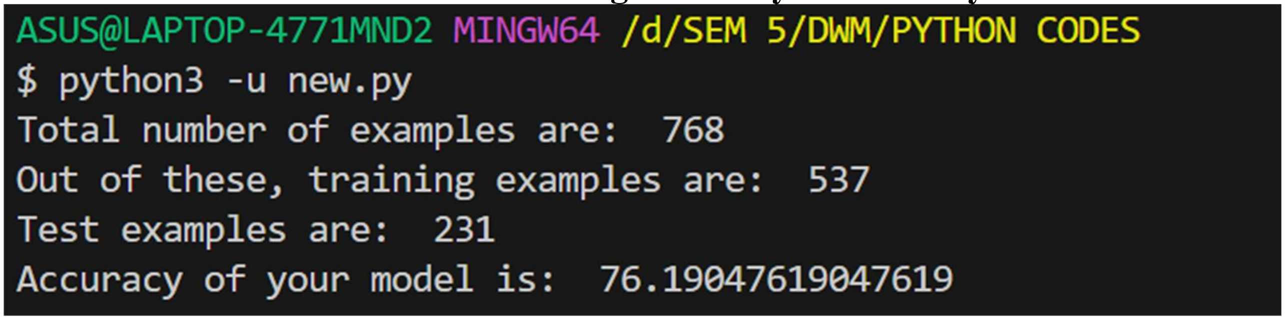
Program Listing and Output:

Source Code:

```
# Importing library
import math
import random
import csv
```

```
# the categorical class names are changed to
numberic data
# eg: yes and no encoded to 1 and 0
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:

    classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata


# Splitting the data
def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    # initially testset will have all the dataset
    test = list(mydata)
    while len(train) < train_num:
        # index generated randomly from range 0
        # to length of testset
        index = random.randrange(len(test))
        # from testset, pop data rows and put it in train
        train.append(test.pop(index))
    return train, test


# Group the data rows under each class yes or
# no in dictionary eg: dict[yes] and dict[no]
def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
        if (mydata[i][-1] not in dict):
            dict[mydata[i][-1]] = []
        dict[mydata[i][-1]].append(mydata[i])
    return dict


# Calculating Mean
def mean(numbers):
    return sum(numbers) / float(len(numbers))

# Calculating Standard Deviation
def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    # eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
    # here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3
    # delete summaries of last class
    del info[-1]
    return info

# find Mean and Standard Deviation under each class
def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info


# Calculate Gaussian Probability Density Function
def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo


# Calculate Class Probabilities
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities


# Make prediction - highest probability is the prediction
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


# returns predictions for a set of examples
def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions

# Accuracy score
def accuracy_rate(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0


# driver code

# add the data path in your system
filename = r'sales.csv'


# load the file and store it in mydata list
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
    mydata[i] = [float(x) for x in mydata[i]]


# split ratio = 0.7
# 70% of data is training data and 30% is test data used for testing
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))

# prepare model
info = MeanAndStdDevForClass(train_data)

# test model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

Output:-



```
ASUS@LAPTOP-4771MND2 MINGW64 /d/SEM 5/DWM/PYTHON CODES
$ python3 -u new.py
Total number of examples are:  768
Out of these, training examples are:  537
Test examples are:  231
Accuracy of your model is:  76.19047619047619
```

Conclusion: Here we implemented the Naive Bayes algorithm