LAB 11 Bayes Classifier

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Task: It is a classification technique based on Bayes" Theorem with an assumption of independence among predictors. In simple terms, 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".

Dataset : Pima-indians-diabetes.csv

It is a classification technique based on Bayes" Theorem with an assumption of independence among predictors. In simple terms, 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".

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

1) Handling Of Data:

- Load the data from the CSV file and split in to training and test data set.
- Training data set can be used to by Naïve Bayes to make predictions.
- And Test data set can be used to evaluate the accuracy of the model.

2) Summarize Data:

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.

- These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.
- Summary data can be break down into the following sub-tasks:

- **Separate Data By Class**: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.
- **Calculate Mean**:We need to calculate the mean of each attribute for a class value.

The mean is the central middle or central tendency of the data, and we will use it as the middle of our gaussian distribution when calculating probabilities.

- **Calculate Standard Deviation**: We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.
- **Summarize Dataset**: For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.
- The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.
- **Summarize Attributes By Class**: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

3) **Make Predictions:**
   - ❖ Making predictions involves calculating the probability that a given data instance belongs to each class,
   - ❖ then selecting the class with the largest probability as the prediction.
   - ❖ Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) **Evaluate Accuracy**: The predictions can be compared to the class values in the test dataset and a classification\ accuracy can be calculated as an accuracy ratio between 0& and 100%.

Naïve Bayes Program:

```
import csv
import random
import math
def safe_div(x,y):
   if y == 0:
      return 0
   return x / y
def loadCsv(filename):
      lines = csv.reader(open(filename, "r"))
      dataset = list(lines)
      for i in range(len(dataset)):
            dataset[i] = [float(x) for x in dataset[i]]
      return dataset

def splitDataset(dataset, splitRatio):
      trainSize = int(len(dataset) * splitRatio)
      trainSet = []
```

```
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]

def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)

    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))

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

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
```

```
            probabilities[classValue] = 1
            for i in range(len(classSummaries)):
                mean, stdev = classSummaries[i]
                x = inputVector[i]
                probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities

def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)

    return predictions

def getAccuracy(testSet, predictions):
   correct = 0
   for i in range(len(testSet)):
     #print(testSet[i][-1]," ",predictions[i])
     if testSet[i][-1] == predictions[i]:
        correct += 1

     return (correct/float(len(testSet))) * 100.0

def main():
    filename = 'pima-indians-diabetes.data.csv'
    splitRatio = 0.67
    dataset = loadCsv(filename)
    trainingSet,testSet=splitDataset(dataset, splitRatio) #dividing into training and test data
    #trainingSet = dataset #passing entire dataset as training data
    #testSet=[[8.0,183.0,64.0,0.0,0.0,23.3,0.672,32.0]]
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingSet), len(testSet)))
    # prepare model
```

```
    summaries = summarizeByClass(trainingSet)
    # test model
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: {0}%'.format(accuracy))

main()
```

Input:

Pima-indians-diabetes.csv

OUTPUT:

```
Split 768 rows into train=576 and test=192 rows
Accuracy: 77.604 %
```

- Naive Bayes is a classification algorithm that is based on Bayes' theorem. It assumes that the features are conditionally independent given the class, which means that the presence or absence of a particular feature does not affect the presence or absence of any other feature.

Here's an example to illustrate how the Naive Bayes algorithm works:

Let's say we have a dataset of emails labeled as either "spam" or "not spam" based on their content. We want to build a Naive Bayes classifier to automatically classify new emails as spam or not spam.

Step 1: Prepare the Data

We preprocess the email data by converting them into numerical features. For example, we might represent each email as a vector of word frequencies or presence indicators.

Step 2: Calculate Class Probabilities

We calculate the probabilities of each class (spam or not spam) based on the frequency of occurrence of each class in the training data.

Step 3: Calculate Feature Probabilities

For each feature (word), we calculate the conditional probability of that feature occurring given each class. This involves counting the frequency of each feature in each class and normalizing it.

Step 4: Calculate the Posterior Probability

Using Bayes' theorem, we calculate the posterior probability of each class given the features of an input email. This is done by multiplying the class probabilities with the product of the conditional feature probabilities.

Step 5: Make a Prediction

We assign the class with the highest posterior probability as the predicted class for the input email. For example, if the posterior probability of spam is higher than not spam, we classify the email as spam.

Step 6: Repeat Steps 4 and 5 for each Input

We repeat steps 4 and 5 for each new email in the test set to obtain predictions for all emails.

Step 7: Evaluate Accuracy

We compare the predicted class labels with the true class labels to evaluate the accuracy of the Naive Bayes classifier.

The Naive Bayes algorithm works well in practice despite its simplifying assumption of feature independence. It is particularly effective when dealing with high-dimensional data, such as text classification tasks. The algorithm is computationally efficient and can handle large datasets efficiently.

- **Step-by-step guide on how to use the Naive Bayes algorithm for classification**

Here's a step-by-step guide on how to use the Naive Bayes algorithm for classification with an example:

Step 1: Import Libraries

First, import the necessary libraries in Python, including scikit-learn, which provides the Naive Bayes algorithm implementation.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Step 2: Prepare the Data

Load your dataset and split it into input features (X) and corresponding class labels (y).

```
# Assuming your data is stored in a pandas DataFrame called 'data'
X = data.drop('target', axis=1)  # Features
y = data['target']  # Class labels
```

Step 3: Split the Data

Split the dataset into training and testing sets. This allows us to train the Naive Bayes algorithm on a portion of the data and evaluate its performance on the remaining unseen data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 4: Create and Train the Naive Bayes Model

Create an instance of the GaussianNB class, which implements the Gaussian Naive Bayes algorithm. Then, train the model using the training data.

```
nb = GaussianNB()
nb.fit(X_train, y_train)
```

Step 5: Make Predictions

Use the trained Naive Bayes model to make predictions on the test set.

```
y_pred = nb.predict(X_test)
```

Step 6: Evaluate Accuracy

Compare the predicted class labels (y_pred) with the true class labels (y_test) to evaluate the accuracy of the Naive Bayes model.

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

You have successfully performed classification using the Naive Bayes algorithm and calculated the accuracy.

Note: The example assumes that you have already preprocessed the dataset and have numerical features. If your data contains categorical or text features, you would need to apply appropriate preprocessing techniques or use variants of Naive Bayes algorithms such as MultinomialNB or BernoulliNB.

Remember to adjust the values according to your specific dataset and requirements. Additionally, you can explore different variants of Naive Bayes algorithms and evaluate their performance on your data.