Name : Apurv Sarode
UID : 2019130054
Subject : AIML
Experiment : 4

Aim : For a given problem statement classify using Naïve Bayes Algorithm.

Theory:

Naïve Bayes

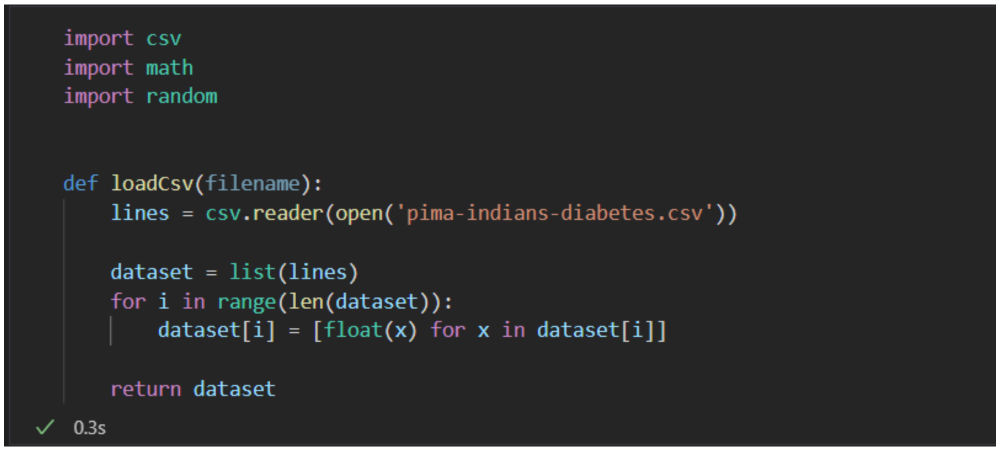
- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of colour, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

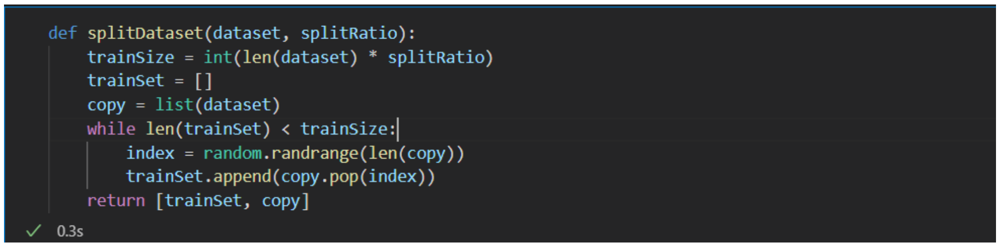
Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Program:



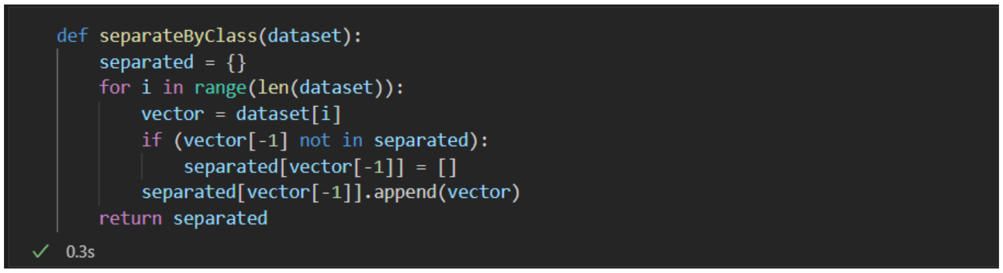
```python
import csv
import math
import random


def loadCsv(filename):
    lines = csv.reader(open('pima-indians-diabetes.csv'))

    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset
```

✓ 0.3s



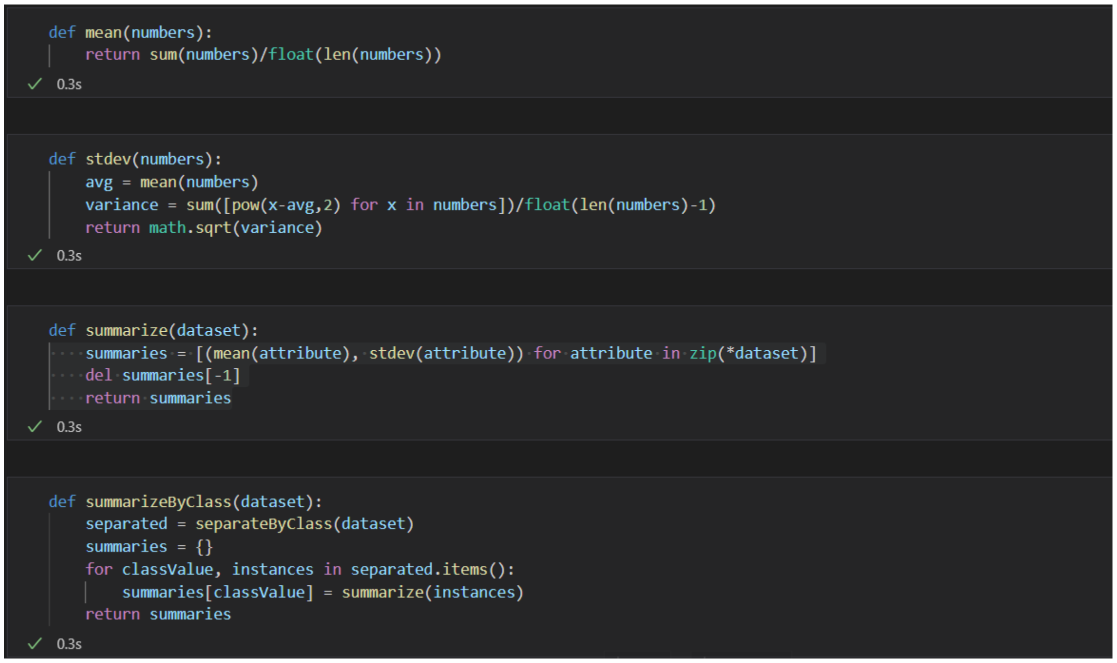
```python
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
```

✓ 0.3s



```python
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
```

✓ 0.3s



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

✓ 0.3s

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

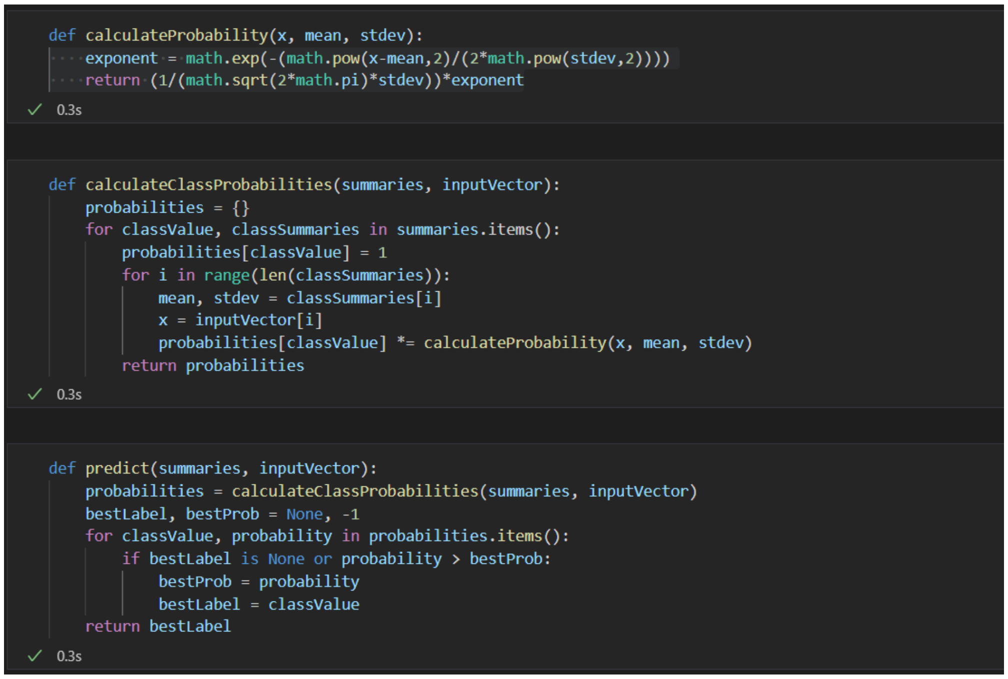
✓ 0.3s

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

✓ 0.3s

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

✓ 0.3s



```python
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
```

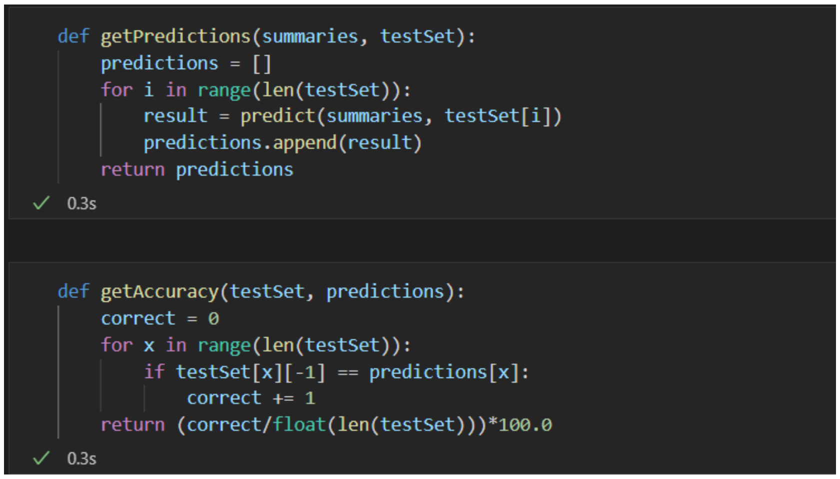
✓ 0.3s

```python
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
```

✓ 0.3s

```python
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
```

✓ 0.3s

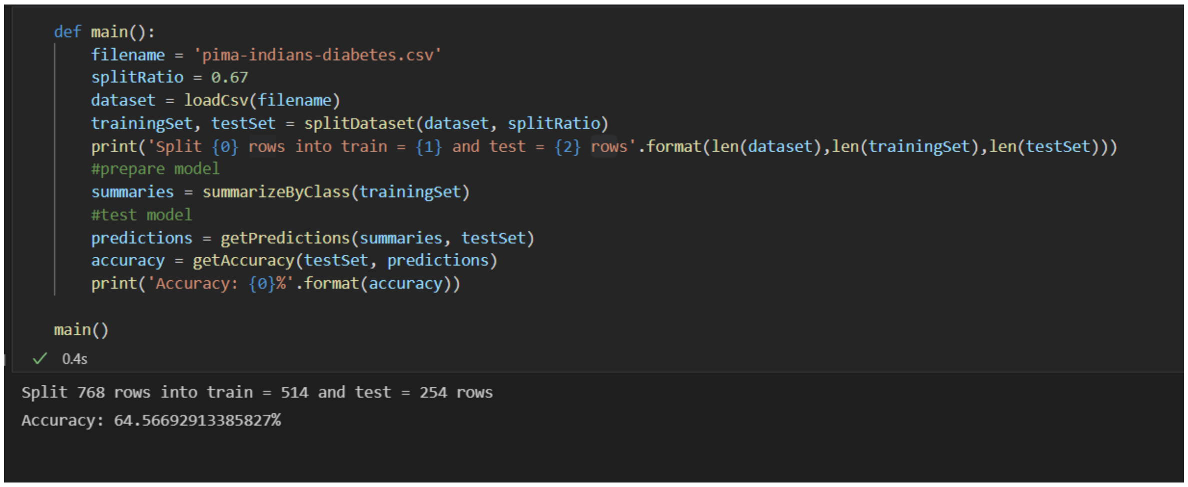


```python
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions
```

✓ 0.3s

```python
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet)))*100.0
```

✓ 0.3s



```python
def main():
    filename = 'pima-indians-diabetes.csv'
    splitRatio = 0.67
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    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()
```

✓ 0.4s

```
Split 768 rows into train = 514 and test = 254 rows
Accuracy: 64.56692913385827%
```

Conclusion:

In the above experiment of AIML Lab, I learnt about Naïve Bayes Algorithm. This algorithm is used for supervised learning models.

The first thing we need to do is load our data file. We can open the file with the open function and read the data lines using the reader

Then we split the data into training and testing dataset.

Then we summarize the training data collected involving 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.

Then make predictions that 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, we define our main function where we call all these methods we have defined, one by one to get the accuracy of the model we have created.