9/11/20

EXPERIMENT NO. 5

1KS17CS030

AIM: Program to implement the naive Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test datasets.

Algorithm:

1. Handling of Data:

- Load the data from the csv file and split into training & test dataset.
- Training dataset can be used to by Naive Beyes to make predictions.
- 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 & 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 breakdown into the following subtasks:
  - 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, 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 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 put it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

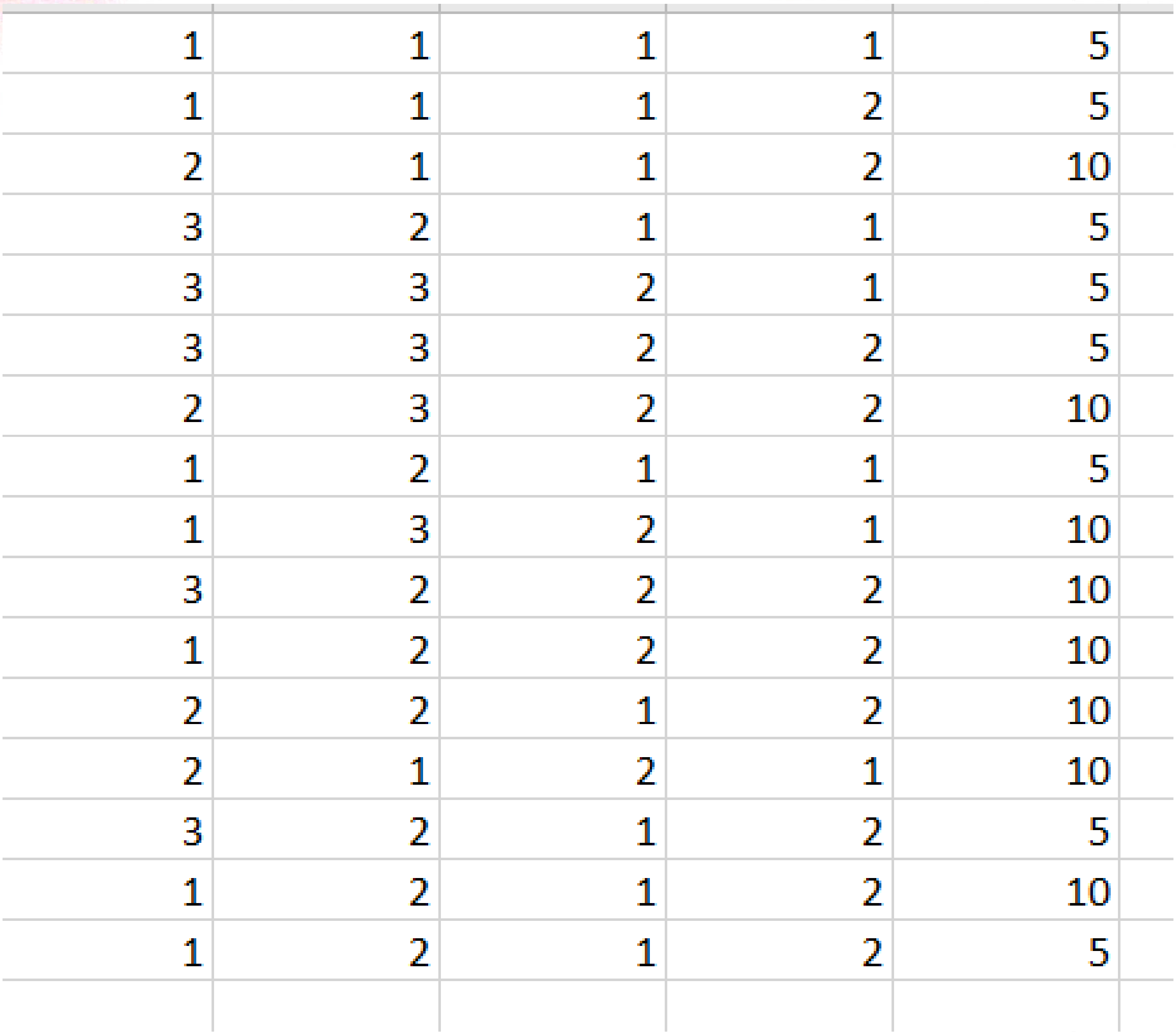
3. Make predictions:

- Making predictions involves calculating the probability that a given data instance belongs to each class
- Select 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 & a classification/accuracy can be calculated as an accuracy ratio between 0 & 100%

Dataset: pima-indians-diabetes.csv.



1	1	1	1	5
1	1	1	2	5
2	1	1	2	10
3	2	1	1	5
3	3	2	1	5
3	3	2	2	5
2	3	2	2	10
1	2	1	1	5
1	3	2	1	10
3	2	2	2	10
1	2	2	2	10
2	2	1	2	10
2	1	2	1	10
3	2	1	2	5
1	2	1	2	10
1	2	1	2	5

Program:

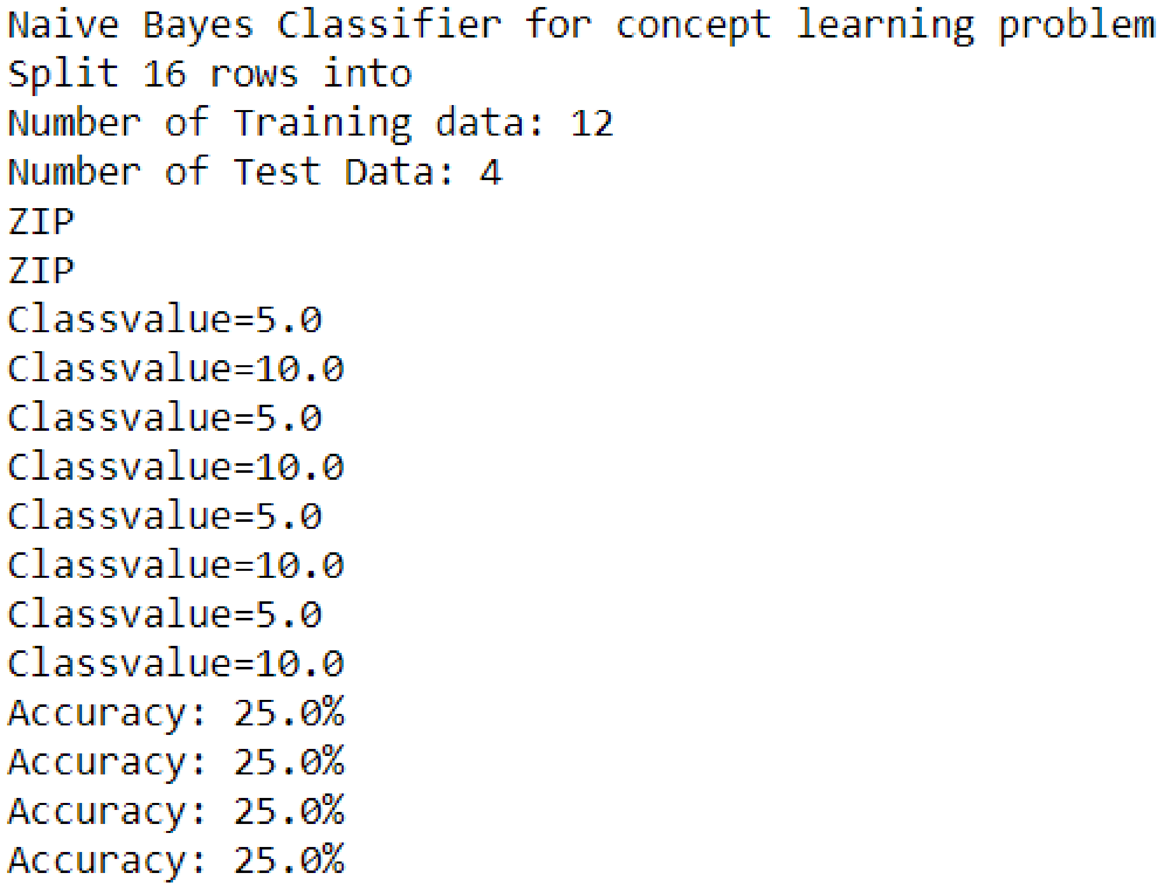
```
import csv
import random
import math
def safe_div (x,y):
    if y==0:
        return 0
    return x/y
def loadCsv (filename):
    lines = csv.reader (open (filename, "r"))
    dataset = list (lines)
    for i in range (len (dataset)):
        dataset[i]= [float(x) for x in dataset[i]]
    return 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):
      separate = 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)):
          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)
  print ('split {0} rows into train= {1} and test= {2} rows. format(
  len(dataset), len(trainingSet), len(testSet)))
```

```
        summaries= summarizeByClass (trainingset)
        predictions = getPredictions (summaries, testset)
        accuracy= getAccuracy (testset, predictions)
        print ('Accuracy: {0} %.'. format (accuracy))

main ()
```

OUTPUT:



```
Naive Bayes Classifier for concept learning problem
Split 16 rows into
Number of Training data: 12
Number of Test Data: 4
ZIP
ZIP
Classvalue=5.0
Classvalue=10.0
Classvalue=5.0
Classvalue=10.0
Classvalue=5.0
Classvalue=10.0
Classvalue=5.0
Classvalue=10.0
Accuracy: 25.0%
Accuracy: 25.0%
Accuracy: 25.0%
Accuracy: 25.0%
```