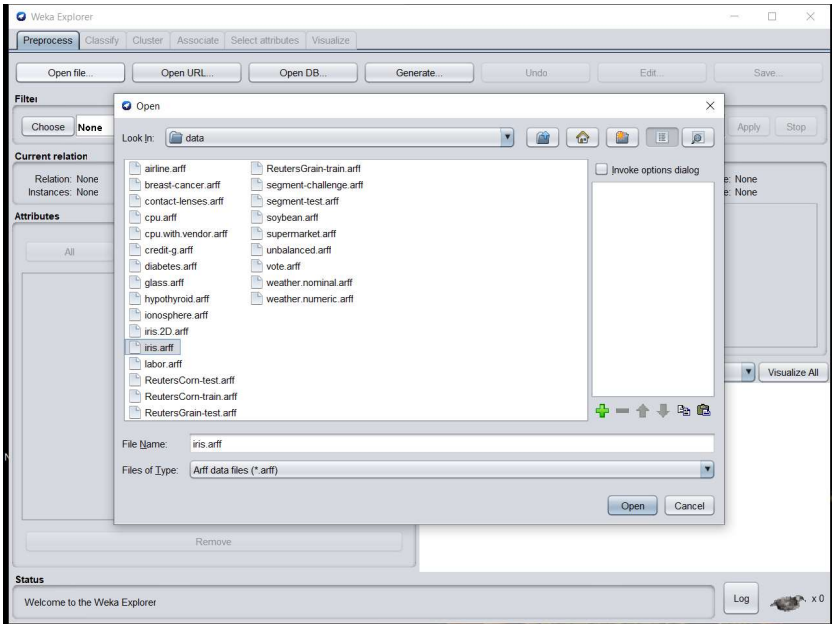
Practical: 10

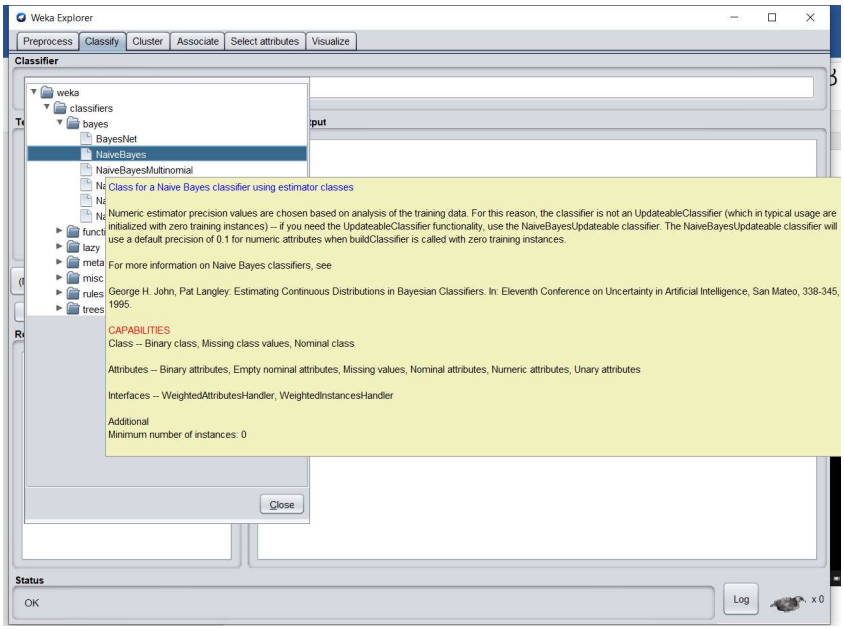
AIM: Implement Naive bayes algorithm in Weka.

Here, we classify the iris flower data.

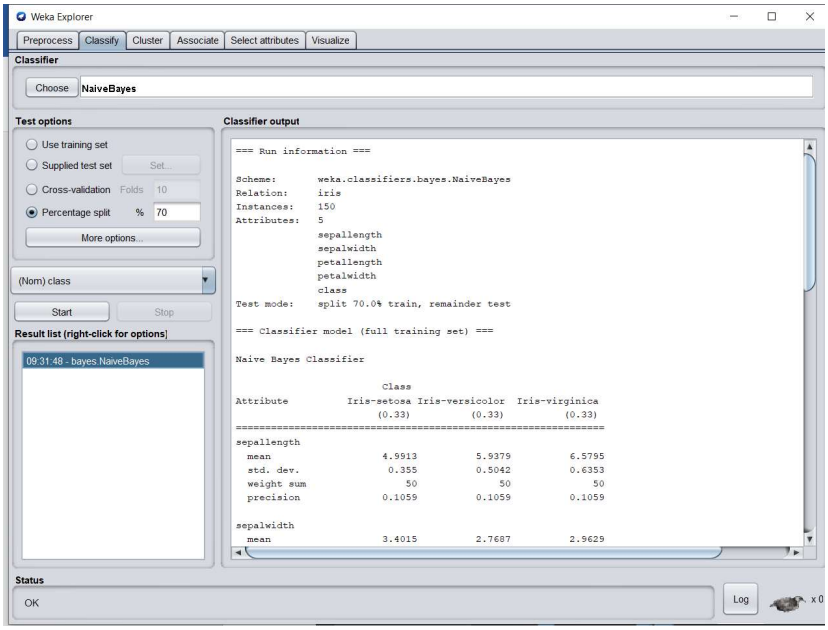
Step 1: To select the dataset from the inbuilt dictionary

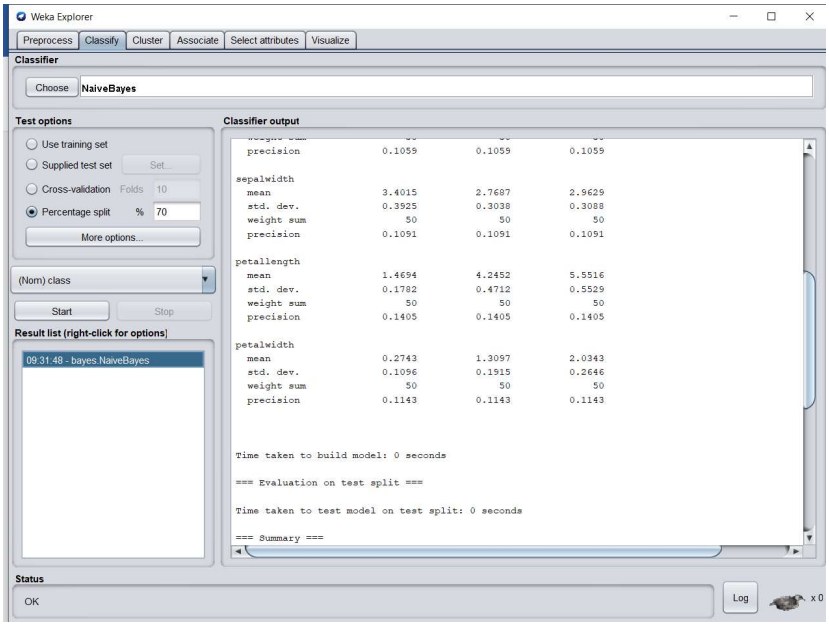


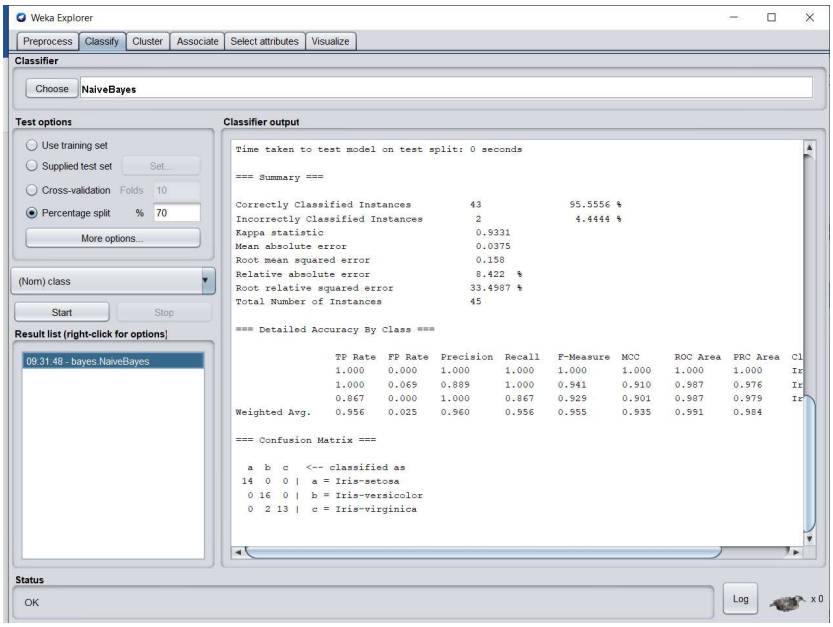
Step 2: We Select Naïve byes for our classification And then hit start button.



Step 3 : in the next picture we got the output.


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
70
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
09:31:48 - bayes.NaiveBayes
Classifier output
=== Run information ===
Scheme: weka.classifiers.bayes.NaiveBayes
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode: split 70.0% train, remainder test
=== Classifier model (full training set) ===
Naive Bayes Classifier
Class
Attribute Iris-setosa Iris-versicolor Iris-virginica
(0.33) (0.33) (0.33)
sepallength
mean 4.9913 5.9379 6.5795
std. dev. 0.355 0.5042 0.6353
weight sum 50 50 50
precision 0.1059 0.1059 0.1059
sepalwidth
mean 3.4015 2.7687 2.9629
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
70
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
09:31:48 - bayes.NaiveBayes
Classifier output
precision 0.1059 0.1059 0.1059
sepalwidth
mean 3.4015 2.7687 2.9629
std. dev. 0.3925 0.3038 0.3088
weight sum 50 50 50
precision 0.1091 0.1091 0.1091
petallength
mean 1.4694 4.2452 5.5516
std. dev. 0.1782 0.4712 0.5529
weight sum 50 50 50
precision 0.1405 0.1405 0.1405
petalwidth
mean 0.2743 1.3097 2.0343
std. dev. 0.1096 0.1915 0.2646
weight sum 50 50 50
precision 0.1143 0.1143 0.1143
Time taken to build model: 0 seconds
=== Evaluation on test split ===
Time taken to test model on test split: 0 seconds
=== Summary ===
Status
OK
Log
x 0



Summary :

Correctly Satisfied Instance is 43 And total instance is 45 so here we get the accuracy is about 95 percent.

Practical: 11

AIM: Implement Naïve bayes classification in java/.net/Python.

```
import numpy as np
from random import randrange
import csv
import math
def load_csv_dataset(filename):
    """Load the CSV file"""
    lines = csv.reader(open(filename, 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]  # Convert String to Float numbers
    return dataset
def mean(numbers):
    """Returns the mean of numbers"""
    return np.mean(numbers)
def stdev(numbers):
    """Returns the std_deviation of numbers"""
    return np.std(numbers)
def sigmoid(z):
    """Returns the sigmoid number"""
    return 1.0 / (1.0 + math.exp(-z))
def cross_validation_split(dataset, n_folds):
    """Split dataset into the k folds. Returns the list of k folds"""
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
```

```
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
def accuracy_metric(actual, predicted):
    """Calculate accuracy percentage"""
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
def evaluate_algorithm(dataset, algorithm, n_folds, ):
    """Evaluate an algorithm using a cross validation split"""
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, )
        actual = [row[-1] for row in fold]
```

```
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
def separate_by_class(dataset):
    """Split training set by class value"""
    separated = {}
    for i in range(len(dataset)):
        row = dataset[i]
        if row[-1] not in separated:
            separated[row[-1]] = []
        separated[row[-1]].append(row)
    return separated
def model(dataset):
    """Find the mean and standard deviation of each feature in dataset"""
    models = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    models.pop() #Remove last entry because it is class value.
    return models
def model_by_class(dataset):
    """find the mean and standard deviation of each feature in dataset by their class"""
    separated = separate_by_class(dataset)
    class_models = {}
    for (classValue, instances) in separated.items():
        class_models[classValue] = model(instances)
    return class_models
def calculate_pdf(x, mean, stdev):
    """Calculate probability using gaussian density function"""
    if stdev == 0.0:
        if x == mean:
```

```
            return 1.0
        else:
            return 0.0
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return 1 / (math.sqrt(2 * math.pi) * stdev) * exponent
def calculate_class_probabilities(models, input):
    """Calculate the class probability for input sample. Combine probability of each feature"""
    probabilities = {}
    for (classValue, classModels) in models.items():
        probabilities[classValue] = 1
        for i in range(len(classModels)):
            (mean, stdev) = classModels[i]
            x = input[i]
            probabilities[classValue] *= calculate_pdf(x, mean, stdev)
    return probabilities
def predict(models, inputVector):
    """Compare probability for each class. Return the class label which has max probability."""
    probabilities = calculate_class_probabilities(models, 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(models, testSet):
    """Get class label for each value in test set."""
    predictions = []
    for i in range(len(testSet)):
```

```
        result = predict(models, testSet[i])
        predictions.append(result)
    return predictions
def naive_bayes(train, test, ):
    """Create a naive bayes model. Then test the model and returns the testing result."""
    summaries = model_by_class(train)
    predictions = getPredictions(summaries, test)
    return predictions
def main():
    # load and prepare data
    filename = 'banknote.csv'
    dataset = load_csv_dataset(filename)
    n_folds = 3
    print ("---------- Gaussian Naive Bayes ----------------")
    accuracy_naive = evaluate_algorithm(dataset, naive_bayes, n_folds)
    print ("Naive Bayes Classification")
    print ('Accuracy in each fold: %s' % accuracy_naive)
    print ('Average Accuracy: %f' % (sum(accuracy_naive) / len(accuracy_naive)))
main()
```

Output:

```
---------- Gaussian Naive Bayes ----------------
Naive Bayes Classification
Accuracy in each fold: [85.55798687089715, 85.77680525164114, 82.27571115973743]
Average Accuracy: 84.536834
```