CSE601: Data Mining
Project 3

Group: 31

Anuj Rastogi – 5013 4324

Nalin Kumar – 5017 0479

Pranshu Pancholi – 5016 9864

Naïve Bayes Classification –

Algorithm Implementation-

We have used Python programming language to implement Naïve Bayes Classifier. The input to algorithm is the dataset file name and the value of K in K fold validation. Below is the pseudo code of the algorithm implementation-

Pseudo Code-

1. Input the dataset file name and value of K for K-fold cross validation.
2. Iterate K times:
    1) ***loadData***(): Load the dataset in a matrix. If the dataset contains 'Present' and 'Absent' attribute values replace 'Present' with 1 and 'Absent' with 0.
    2) ***splitData():*** Randomly select (K-1)/K parts of the dataset as Training Data and the remaining 1/K part as Test Data.
    3) ***createLabelMap():*** Create a map with keys 0 and 1 for the two labels. Key 0 contains the list of lists of training samples with label 0. Key 1 contains the list of lists of training samples with label 1.
    4) ***calculateMeanSD():*** Create a map called 'meanVarMap' with keys 0 and 1 representing 2 labels. Key 0 and key 1 will contain value as array of [mean, standard deviation]. Here mean and standard deviation are lists of mean and standard deviation of all attributes for respective labels.
    To calculate mean of each attribute values with label 1 we sum all the attribute values with label 1 and divide it by the number of samples with label 1. Similarly, we calculate mean of each attribute values with label 0. Thus, the mean array contains the mean of all the attributes in the dataset.
    To calculate the standard deviation of each attribute we use the below formula. After calculating mean and standard deviation we make a pair and put it in the map for respective label.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

where, μ = Mean of attribute

x_i = attribute value

σ = Standard Deviation

N = Total no. of samples

    5) ***predictLabel():*** We have used Normal Distribution to calculate the probability that each test sample belongs to a particular class. Here, we predict the label of each test sample from the test dataset. To predict the

label we have calculated the class probabilities of the sample. A label is assigned to the sample which has higher probability. We have used the following formula to calculate probability using normal distribution –

$$P(A_i \mid c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(A_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

where, P(Ai|Cj) = Probability that a sample Ai belongs to a class cj

σ ij = Standard Deviation

μ = Mean

6) Final step is to calculate different measures like accuracy, precision, recall and F-Measure. For this we calculate True Positive, True Negative, False Positive and False Negative by comparing predicted and actual label array. We store each measure value for the Kth iteration in respective measure array.

3. Finally we calculate the average of K values of accuracy, precision, recall, F-Measure and output the resultant measure values.

Experiment Results –

a) Dataset – 'project3_dataset1.txt'

1) **Without** K-Fold Cross Validation

Accuracy	96.4912280
Precision	100.0
Recall	90.4761904
F-Measure	95.0

Fig a. Measure values for Dataset1

2) **With** K-Fold Cross Validation (K = 10)

Accuracy	94.9122807018
Precision	92.2778032037
Recall	93.9387191756
F-Measure	92.9079910438

Fig b. Measure values for Dataset1

```
Enter the dataset file name: project3_dataset1.txt
Enter Integer value of K: 10
Average Accuracy 94.9122807018
Average Precision 92.2778032037
Average Recall 93.9387191756
Average FMeasure 92.9079910438
```

b) Dataset – 'project3_dataset2.txt'

1) **Without** K-Fold Cross Validation

Accuracy	74.4680851
Precision	64.7058823
Recall	64.7058823
F-Measure	64.7058823

Fig c. Measure values for Dataset2

2) **With** K-Fold Cross Validation (K = 10)

Accuracy	73.6170212766
Precision	65.9541341012
Recall	60.810828877
F-Measure	62.3605015674

Fig d. Measure values for Dataset2

```
Enter the dataset file name: project3_dataset2.txt
Enter Integer value of K: 10
Average Accuracy 73.6170212766
Average Precision 65.9541341012
Average Recall 60.810828877
Average FMeasure 62.3605015674
```

Choice Description-

Continuous Features – We used normal distribution of the data to deal with continuous features. First we calculated the mean and standard deviation of the training dataset, then we predicted the labels of the test data by calculating the posterior probability using normal probability distribution function. The label with greater posterior probability is the predicted label for that sample.

Zero Probability – We used Laplacian Correction to deal with zero probability issue. To achieve this we add 1 to each attribute value in case we calculate a zero probability. By doing this we are assuming that our training set is very large, so large that adding one to each count would only make a negligible difference in the estimated probabilities, therefore would avoid the case of zero probability values

Result Analysis-

By using K-Fold cross validation we observed that by using K-fold we achieved much more stable measure values. This is evident from the experiment result above. Also, just the accuracy cannot be the best measure alone as model may predict all the specific class in dataset and achieve maximum accuracy although. Thus, considering other

measures like precision with represents exactness and recall which represents completeness of a model and F1 Measure which gives a balance between the 2 are also important measure to interpret a classifier model. We also observed that measure values for dataset2 is lower than dataset1. This is because we used probability distribution function which works best for continuous data.

Pros-

3) It is simple to understand and implement.
4) Efficient to run on larger datasets.
5) Comparable in performance to decision trees.
6) It is a statistical classifier and predicts class membership probabilities.
7) Fast to train and classify.
8) Not sensitive to irrelevant features.
9) Handles both real and discrete data values.

Cons-

1) It assumes independence of features which may sometimes lead to misclassification.

K Nearest Neighbors Classification

Introduction

K-Nearest Neighbors algorithm is a non-parametric method widely used for classification and regression. In both cases, input consists of the K closest examples in the feature space to the test data point. In a typical K-NN Classification algorithm, the output is basically a class membership which implies the fact that all the data points should belong to a particular class. A new test data point is typically classified by a majority vote of its neighbors, with the test data point being assigned to a class most amongst its k nearest neighbors, where k is a positive number typically kept small. However, if K=1, then the test sample data point is simply assigned to the class of that single nearest neighbor.

Since as already stated above, K-NN is a non-parametric lazy learning algorithm, it means that it does not make any assumptions on the underlying data distribution. This is most useful in real world scenarios where most of the practical data does not obey the typical theoretical assumptions made. By the term lazy, it is meant to convey that the algorithm does not doesn't use the training data points to perform any kind of generalizations and there is no explicit training phase which is typically uncommon in other types of classification algorithms. This essentially means that the training phase is very fast. However, lack of generalization means that K-NN keeps all of the training data to perform predictions in the testing phase which can be a cause of concern since it expensive in terms of time, memory and cost. Another shortcoming of the K-NN algorithm is that is sensitive to the local structure of the data which will be discussed in later subsections.

K-NN Algorithm Flow

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, *k* is a user-defined constant, and an unlabelled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman. Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbourer Neighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbours due to

their large number. One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbours. The class of each of the *k* nearest points is typically multiplied by a weight proportional to the inverse of the distance from that point to the test point.

Assumptions in K-NN

- **Data Points –** K-NN assumes that the data is in a feature space. More precisely, the data points are in a metric space. The data points can be scalars or possibly even be multidimensional vectors. Since the points are in feature space, there is a notion of distance widely being used in this algorithm such as Euclidean Distance for continuous features, Hamming's distance for categorical features etc
- **Class Labels –** Each of the training data consists of a set of vectors called as features or attributes and class labels associated with each vector which precisely defines the class to which that particular data point belongs to. In the simplest case there might be 2 classes, but K-NN works equally well for arbitrary number of classes
- **Parametric Selection –** K is the number of nearest neighbours in this algorithm which defines the number of neighbours that should be used to consider while assigning a class label to an unlabelled test sample data point. Other aspects regarding this parameter are described more clearly in the choice description subsection below.

Choice Description

K – Number of Nearest Neighbours

The best choice of *k* depends upon the data. Generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good *k* can be selected by various heuristic techniques such as hyper parameter optimization. The special case where the class is predicted to be the class of the closest training sample (i.e. when $k = 1$) is called the nearest neighbour algorithm. The accuracy of the *k*-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features by the mutual information of the training data with the training classes. In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal *k* in this setting is via bootstrap method

Distance Measures

Various distance measures can be used to compute the distance between test sample data point and training data points depending upon the type of features. In our case for this

project, we only encountered continuous as well as categorical features. Based on this observation, we calculated the Euclidean distance between features that are of continuous type and Hamming's distance between features that are of categorical type and eventually added the two partial distance results to compute the final total distance between test sample data point and all other training data points in order to accurately predict its label.

Continuous Features/Attributes

There were both continuous features as well as categorical features present in the given datasets in the project. In order to compute distance between two data points, we calculated Euclidean distance between the corresponding continuous features of both the data points.

Categorical Features/Attributes

There were both continuous features as well as categorical features present in the given datasets in the project. In order to compute distance between two data points, we calculated Hamming's distance between the corresponding categorical features of both the data points. For this, we took the distance between two categorical features to be 1 when the features were different after ignoring the case and 0 when the two categorical features were equal.

Result Analysis

Algorithm/Methodology Description

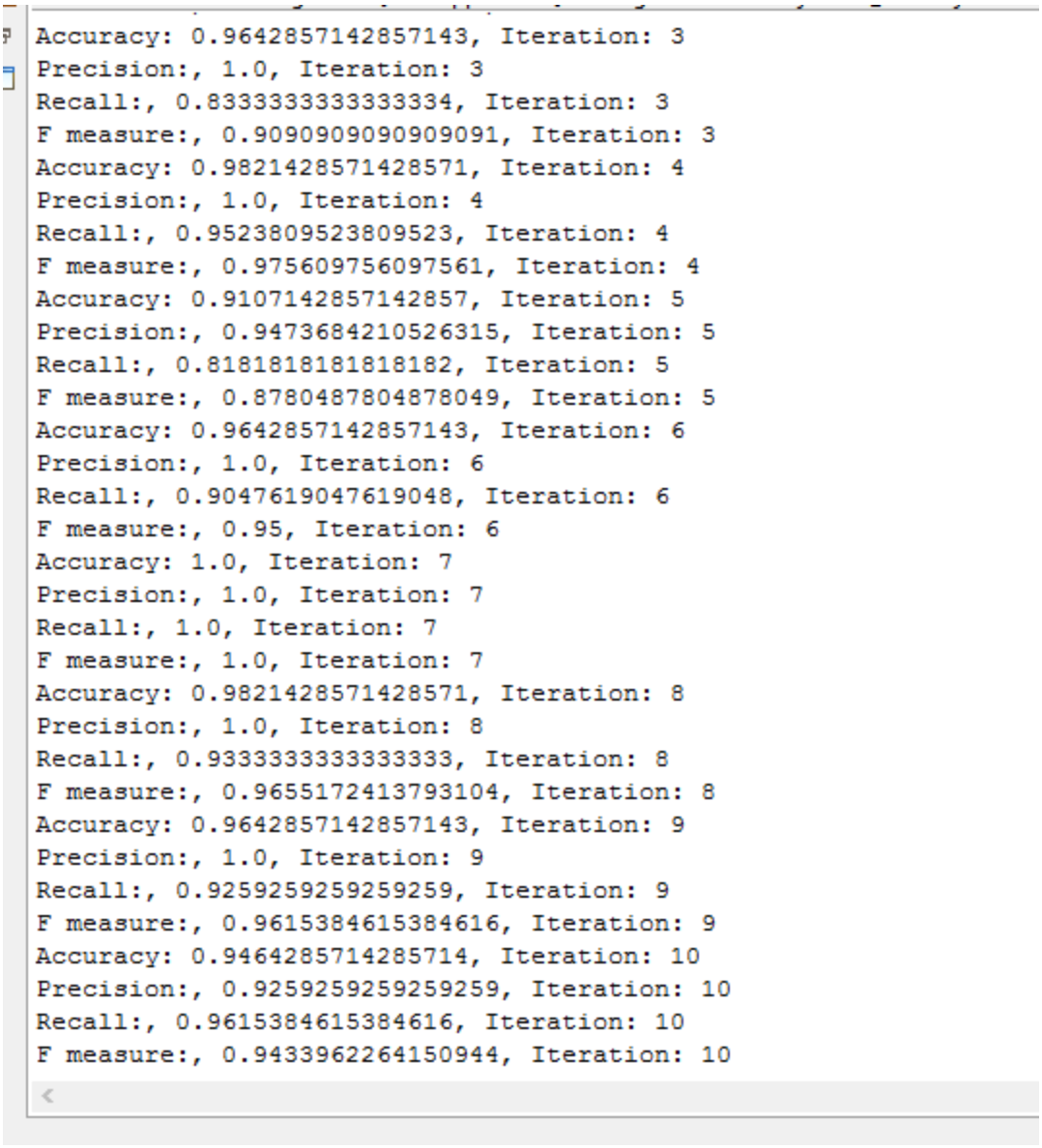
Initially, we stored all the configurable properties inside a single configuration properties file in order to make our code robust and scalable with respect to certain input parameters such as filename, input K value, no of folds etc. Following this, we followed an object oriented approach and saved all the relevant data in-memory in most efficient data structures. Initial data is first sampled based on two methods. In the first method, we adopted 10-fold cross validation method to partition training and test data and ran the algorithm for 10 combinations of training and test data. In the second approach, we randomly shuffled the data, split the data into two parts based on input fold value and ran the algorithm only once to visualize results in all the scenarios. Subsequently, we iterated over all test data points and evaluated their distances from all the training points to get the K Nearest Neighbours. Following this, we implemented a distance based comparator which sorts distances of all training points to a particular test sample data point and pick the top K training data points which have minimum distance with respect to the test sample data point. Finally, we employ two types of voting schemes to assign the class label to unknown test sample data point. In the first approach, we use majority voting scheme in which we pick the label with the highest number of votes amongst the K nearest neighbours. In the second approach, we evaluate the weighted vote of all the labels based on the inverse of square of the distances between that particular nearest training data point and test sample data point.

Pre-processing

For pre-processing, initially we normalize input data based on all the continuous features of all the data points since we understand that attributes need to be scaled since to prevent the distance measures being dominated by one of the attributes possessing a larger value in comparison to other attributes. Subsequently, we split the input data into training and test data based on two approaches, namely 10-fold cross validation and random shuffling as explained above. We followed this approach for two reasons, namely to avoid overfitting and to visualize how accuracy is improved by choosing different kinds of training and test data in multiple runs of K-Nearest Algorithm

Results

K=3, Dataset1, 10-Fold Cross Validation (Majority Voting)



K=10, Dataset1, 10-Fold Cross Validation (Majority Voting)

```
Precision:, 1.0, Iteration: 3
Recall:, 0.9166666666666666, Iteration: 3
F measure:, 0.9565217391304348, Iteration: 3
Accuracy: 0.9821428571428571, Iteration: 4
Precision:, 1.0, Iteration: 4
Recall:, 0.9523809523809523, Iteration: 4
F measure:, 0.975609756097561, Iteration: 4
Accuracy: 0.9464285714285714, Iteration: 5
Precision:, 1.0, Iteration: 5
Recall:, 0.8636363636363636, Iteration: 5
F measure:, 0.926829268292683, Iteration: 5
Accuracy: 0.9642857142857143, Iteration: 6
Precision:, 1.0, Iteration: 6
Recall:, 0.9047619047619048, Iteration: 6
F measure:, 0.95, Iteration: 6
Accuracy: 1.0, Iteration: 7
Precision:, 1.0, Iteration: 7
Recall:, 1.0, Iteration: 7
F measure:, 1.0, Iteration: 7
Accuracy: 0.9821428571428571, Iteration: 8
Precision:, 1.0, Iteration: 8
Recall:, 0.9333333333333333, Iteration: 8
F measure:, 0.9655172413793104, Iteration: 8
Accuracy: 0.9642857142857143, Iteration: 9
Precision:, 1.0, Iteration: 9
Recall:, 0.9259259259259259, Iteration: 9
F measure:, 0.9615384615384616, Iteration: 9
Accuracy: 0.9285714285714286, Iteration: 10
Precision:, 0.9583333333333334, Iteration: 10
Recall:, 0.8846153846153846, Iteration: 10
F measure:, 0.92, Iteration: 10
```

K=20, Dataset1, 10-Fold Cross Validation (Majority Voting)

```
Accuracy: 0.9821428571428571, Iteration: 3
Precision:, 1.0, Iteration: 3
Recall:, 0.9166666666666666, Iteration: 3
F measure:, 0.9565217391304348, Iteration: 3
Accuracy: 0.9821428571428571, Iteration: 4
Precision:, 1.0, Iteration: 4
Recall:, 0.9523809523809523, Iteration: 4
F measure:, 0.975609756097561, Iteration: 4
Accuracy: 0.9285714285714286, Iteration: 5
Precision:, 1.0, Iteration: 5
Recall:, 0.8181818181818182, Iteration: 5
F measure:, 0.9, Iteration: 5
Accuracy: 0.9821428571428571, Iteration: 6
Precision:, 1.0, Iteration: 6
Recall:, 0.9523809523809523, Iteration: 6
F measure:, 0.975609756097561, Iteration: 6
Accuracy: 1.0, Iteration: 7
Precision:, 1.0, Iteration: 7
Recall:, 1.0, Iteration: 7
F measure:, 1.0, Iteration: 7
Accuracy: 0.9642857142857143, Iteration: 8
Precision:, 0.9333333333333333, Iteration: 8
Recall:, 0.9333333333333333, Iteration: 8
F measure:, 0.9333333333333333, Iteration: 8
Accuracy: 0.9285714285714286, Iteration: 9
Precision:, 1.0, Iteration: 9
Recall:, 0.8518518518518519, Iteration: 9
F measure:, 0.92, Iteration: 9
Accuracy: 0.9285714285714286, Iteration: 10
Precision:, 0.9583333333333334, Iteration: 10
Recall:, 0.8846153846153846, Iteration: 10
F measure:, 0.92, Iteration: 10
```

K=3, Dataset2, 10-Fold Cross Validation (Majority Voting)

```
Precision:, 0.6428571428571429, Iteration: 3
Recall:, 0.47368421052631576, Iteration: 3
F measure:, 0.5454545454545454, Iteration: 3
Accuracy: 0.7608695652173914, Iteration: 4
Precision:, 0.6428571428571429, Iteration: 4
Recall:, 0.6, Iteration: 4
F measure:, 0.6206896551724138, Iteration: 4
Accuracy: 0.6086956521739131, Iteration: 5
Precision:, 0.6666666666666666, Iteration: 5
Recall:, 0.2857142857142857, Iteration: 5
F measure:, 0.4, Iteration: 5
Accuracy: 0.6086956521739131, Iteration: 6
Precision:, 0.5, Iteration: 6
Recall:, 0.3888888888888889, Iteration: 6
F measure:, 0.4375, Iteration: 6
Accuracy: 0.717391304347826, Iteration: 7
Precision:, 0.5, Iteration: 7
Recall:, 0.3076923076923077, Iteration: 7
F measure:, 0.38095238095238093, Iteration: 7
Accuracy: 0.8478260869565217, Iteration: 8
Precision:, 0.6666666666666666, Iteration: 8
Recall:, 0.6, Iteration: 8
F measure:, 0.631578947368421, Iteration: 8
Accuracy: 0.5869565217391305, Iteration: 9
Precision:, 0.36363636363636365, Iteration: 9
Recall:, 0.25, Iteration: 9
F measure:, 0.2962962962962963, Iteration: 9
Accuracy: 0.7608695652173914, Iteration: 10
Precision:, 0.6153846153846154, Iteration: 10
Recall:, 0.5714285714285714, Iteration: 10
F measure:, 0.5925925925925926, Iteration: 10
```

K=10, Dataset2, 10-Fold Cross Validation (Majority Voting)

```
Accuracy: 0.6956521739130435, Iteration: 3
Precision:, 0.7777777777777778, Iteration: 3
Recall:, 0.3684210526315789, Iteration: 3
F measure:, 0.5, Iteration: 3
Accuracy: 0.717391304347826, Iteration: 4
Precision:, 0.6, Iteration: 4
Recall:, 0.4, Iteration: 4
F measure:, 0.48, Iteration: 4
Accuracy: 0.4782608695652174, Iteration: 5
Precision:, 0.0, Iteration: 5
Recall:, 0.0, Iteration: 5
F measure:, 0.0, Iteration: 5
Accuracy: 0.6086956521739131, Iteration: 6
Precision:, 0.5, Iteration: 6
Recall:, 0.1111111111111111, Iteration: 6
F measure:, 0.18181818181818182, Iteration: 6
Accuracy: 0.7608695652173914, Iteration: 7
Precision:, 0.75, Iteration: 7
Recall:, 0.23076923076923078, Iteration: 7
F measure:, 0.35294117647058826, Iteration: 7
Accuracy: 0.8043478260869565, Iteration: 8
Precision:, 0.5714285714285714, Iteration: 8
Recall:, 0.4, Iteration: 8
F measure:, 0.47058823529411764, Iteration: 8
Accuracy: 0.6739130434782609, Iteration: 9
Precision:, 0.5714285714285714, Iteration: 9
Recall:, 0.25, Iteration: 9
F measure:, 0.34782608695652173, Iteration: 9
Accuracy: 0.6956521739130435, Iteration: 10
Precision:, 0.5, Iteration: 10
Recall:, 0.14285714285714285, Iteration: 10
F measure:, 0.2222222222222222, Iteration: 10
```

K=25, Dataset2, 10-Fold Cross Validation (Majority Voting)

```
Precision:, 0.7777777777777778, Iteration: 3
Recall:, 0.3684210526315789, Iteration: 3
F measure:, 0.5, Iteration: 3
Accuracy: 0.717391304347826, Iteration: 4
Precision:, 0.6, Iteration: 4
Recall:, 0.4, Iteration: 4
F measure:, 0.48, Iteration: 4
Accuracy: 0.5652173913043478, Iteration: 5
Precision:, 0.6666666666666666, Iteration: 5
Recall:, 0.09523809523809523, Iteration: 5
F measure:, 0.16666666666666666, Iteration: 5
Accuracy: 0.6521739130434783, Iteration: 6
Precision:, 0.6666666666666666, Iteration: 6
Recall:, 0.2222222222222222, Iteration: 6
F measure:, 0.3333333333333333, Iteration: 6
Accuracy: 0.8043478260869565, Iteration: 7
Precision:, 0.8333333333333334, Iteration: 7
Recall:, 0.38461538461538464, Iteration: 7
F measure:, 0.5263157894736842, Iteration: 7
Accuracy: 0.8260869565217391, Iteration: 8
Precision:, 0.625, Iteration: 8
Recall:, 0.5, Iteration: 8
F measure:, 0.5555555555555556, Iteration: 8
Accuracy: 0.6304347826086957, Iteration: 9
Precision:, 0.4444444444444444, Iteration: 9
Recall:, 0.25, Iteration: 9
F measure:, 0.32, Iteration: 9
Accuracy: 0.7391304347826086, Iteration: 10
Precision:, 0.6666666666666666, Iteration: 10
Recall:, 0.2857142857142857, Iteration: 10
F measure:, 0.4, Iteration: 10
```

K=3, Dataset2, Random Sampling/Shuffling (Majority Voting)

```
416
46
Accuracy: 0.7391304347826086, Iteration: 1
Precision:, 0.6, Iteration: 1
Recall:, 0.6, Iteration: 1
F measure:, 0.6, Iteration: 1
```

K=10, Dataset2, Random Sampling/Shuffling (Majority Voting)

```
416
46
Accuracy: 0.8478260869565217, Iteration: 1
Precision:, 0.6666666666666666, Iteration: 1
Recall:, 0.7272727272727273, Iteration: 1
F measure:, 0.6956521739130435, Iteration: 1
```

K=25, Dataset2, Random Sampling/Shuffling (Majority Voting)

```
416
46
Accuracy: 0.6086956521739131, Iteration: 1
Precision:, 0.6666666666666666, Iteration: 1
Recall:, 0.2, Iteration: 1
F measure:, 0.3076923076923077, Iteration: 1
```

K=3, Dataset1, Random Sampling/Shuffling (Majority Voting)

```
513
56
Accuracy: 0.9821428571428571, Iteration: 1
Precision:, 1.0, Iteration: 1
Recall:, 0.9333333333333333, Iteration: 1
F measure:, 0.9655172413793104, Iteration: 1
```

K=10, Dataset1, Random Sampling/Shuffling (Majority Voting)

```
513
56
Accuracy: 1.0, Iteration: 1
Precision:, 1.0, Iteration: 1
Recall:, 1.0, Iteration: 1
F measure:, 1.0, Iteration: 1
```

K=25, Dataset1, Random Sampling/Shuffling (Majority Voting)

```
513
56
Accuracy: 0.9464285714285714, Iteration: 1
Precision:, 1.0, Iteration: 1
Recall:, 0.8125, Iteration: 1
F measure:, 0.896551724137931, Iteration: 1
```

K=10, Dataset1, 10-Fold Cross Validation (Weighted Voting)

```
Precision:, 1.0, Iteration: 3
Recall:, 0.9166666666666666, Iteration: 3
F measure:, 0.9565217391304348, Iteration: 3
Accuracy: 0.9821428571428571, Iteration: 4
Precision:, 1.0, Iteration: 4
Recall:, 0.9523809523809523, Iteration: 4
F measure:, 0.975609756097561, Iteration: 4
Accuracy: 0.9464285714285714, Iteration: 5
Precision:, 0.9523809523809523, Iteration: 5
Recall:, 0.9090909090909091, Iteration: 5
F measure:, 0.9302325581395349, Iteration: 5
Accuracy: 0.9642857142857143, Iteration: 6
Precision:, 1.0, Iteration: 6
Recall:, 0.9047619047619048, Iteration: 6
F measure:, 0.95, Iteration: 6
Accuracy: 1.0, Iteration: 7
Precision:, 1.0, Iteration: 7
Recall:, 1.0, Iteration: 7
F measure:, 1.0, Iteration: 7
Accuracy: 0.9821428571428571, Iteration: 8
Precision:, 1.0, Iteration: 8
Recall:, 0.9333333333333333, Iteration: 8
F measure:, 0.9655172413793104, Iteration: 8
Accuracy: 0.9642857142857143, Iteration: 9
Precision:, 1.0, Iteration: 9
Recall:, 0.9259259259259259, Iteration: 9
F measure:, 0.9615384615384616, Iteration: 9
Accuracy: 0.9464285714285714, Iteration: 10
Precision:, 0.96, Iteration: 10
Recall:, 0.9230769230769231, Iteration: 10
F measure:, 0.9411764705882353, Iteration: 10
```

K=10, Dataset2, 10-Fold Cross Validation (Weighted Voting)

```
Precision:, 0.8333333333333334, Iteration: 3
Recall:, 0.5263157894736842, Iteration: 3
F measure:, 0.6451612903225806, Iteration: 3
Accuracy: 0.6739130434782609, Iteration: 4
Precision:, 0.5, Iteration: 4
Recall:, 0.4, Iteration: 4
F measure:, 0.4444444444444444, Iteration: 4
Accuracy: 0.5434782608695652, Iteration: 5
Precision:, 0.5, Iteration: 5
Recall:, 0.14285714285714285, Iteration: 5
F measure:, 0.2222222222222222, Iteration: 5
Accuracy: 0.6086956521739131, Iteration: 6
Precision:, 0.5, Iteration: 6
Recall:, 0.2222222222222222, Iteration: 6
F measure:, 0.3076923076923077, Iteration: 6
Accuracy: 0.7391304347826086, Iteration: 7
Precision:, 0.5714285714285714, Iteration: 7
Recall:, 0.3076923076923077, Iteration: 7
F measure:, 0.4, Iteration: 7
Accuracy: 0.8043478260869565, Iteration: 8
Precision:, 0.5714285714285714, Iteration: 8
Recall:, 0.4, Iteration: 8
F measure:, 0.47058823529411764, Iteration: 8
Accuracy: 0.6304347826086957, Iteration: 9
Precision:, 0.42857142857142855, Iteration: 9
Recall:, 0.1875, Iteration: 9
F measure:, 0.2608695652173913, Iteration: 9
Accuracy: 0.6956521739130435, Iteration: 10
Precision:, 0.5, Iteration: 10
Recall:, 0.2857142857142857, Iteration: 10
F measure:, 0.36363636363636365, Iteration: 10
```

K=10, Dataset1, Random Shuffling (Weighted Voting)

```
513
56
Accuracy: 0.9642857142857143, Iteration: 1
Precision:, 1.0, Iteration: 1
Recall:, 0.9259259259259259, Iteration: 1
F measure:, 0.9615384615384616, Iteration: 1
```

K=10, Dataset2, Random Shuffling (Weighted Voting)

```
416
46
Accuracy: 0.6739130434782609, Iteration: 1
Precision:, 0.5, Iteration: 1
Recall:, 0.26666666666666666, Iteration: 1
F measure:, 0.34782608695652173, Iteration: 1
```

Result Analysis/Observations – Pros and Cons of Approaches

- **Measures** – As we know accuracy is not a good measure of correct classification since accuracy might be misleading in case examples of one class severely dominates examples of other classes. Henceforth, we relied upon F-Measure to judge correctness of our results since this measure combines the goodness of two measures in itself namely, Precision and Recall.
- **Increasing the value of K** – As we increased K for both 10-fold cross validation and random shuffling approaches based on different datasets, F-Measures and other measures as well increased marginally while at the cost of doing more computation. Henceforth, we believe that increasing the value of K after a certain value, say 10, is not a good approach, especially when the dataset size is not very large.
- **Datasets 1 and 2** – We can clearly comprehend from the results that all the measures gave excellent results in case of dataset1 going till 100% F-Measure in certain cases, while in case of dataset2, we got poor results in terms of all the measures. Henceforth, we can deduce that there might be the presence of outliers or noise points in dataset2 which is contributing towards these discriminating results
- **10-Fold Cross Validation vis a vis Random Shuffling** – As we can visualize from the results above, we can say that results from dataset1 in case of 10-fold cross validation approach versus random data shuffling approach were almost the same. However, for dataset2, we got better results for F-Measure and other measures in case of random shuffling approach versus 10-fold approach. This leads us to the idea that random shuffling approach based on dividing the data into training data and test data randomly from input data might be a better approach for such datasets containing noisy/outlier points. However, 10-fold cross validation is a better approach since by partitioning the input data in several combinations of training and test data sets, leads us to visualize and choose the best portioning scheme and allows us to comprehend the best possible choice of training dataset.
- **Weighted Voting vis a vis Majority Voting** – We implemented both the voting approaches into our algorithm and tried to compare the results. As can be seen from the results above, we can see that voting algorithm doesn't influence the results in our case since the results came out to be similar for both the datasets for both the voting algorithms. Although we can comprehend that in case of large sized datasets

and in cases where there are multiple datasets, weighted voting scheme is a much better approach since it gives a vote to every class label based on distance between training and test data points and is more fair as opposed to majority voting. This might be a problem in the case of large datasets and multiple classes since there might be a case where there is one class which contains several training examples but the points itself are far away from the training data points

- **Choosing the value of K** – If K is too small, then the K-NN algorithm becomes highly susceptible to noisy/outlier points and if K is kept too large then neighborhood may include points from other classes and time complexity will be large at the cost of marginal increase in accuracy, precision, recall and F-measure. Henceforth, utmost care must be taken while choosing the value of K. In our opinion, ideally, K should be a number close to the square root or the cube root of the input data size since this value is then never too small and neither too large. In such a case, once can reap the benefits of both the cases while avoiding the disadvantages.

Decision Tree

A decision tree is a classification model. This model classifies the data by directing the data in a particular direction according to a tree that has already been learnt. The learnt tree has nodes that signifies each attribute of the data. Each node has some value, which tells us which way the data should go. This value is called the split value and is determined by following several methods and calculating some indexes. There are three common indexes that can be used:

a. Gini Index
b. Entropy
c. Miss classification Error

The best split for any attribute is one which has the minimum impurity or maximum homogeneity or minimum Gini Index.

Now, the attributes to classify can be of three types:

a. Ordinal
b. Continuous
c. Nominal

Hence, the split can be done based on the type of attribute. Also, there can be many number of splits for a node. Based on the strategy for split, tree can be consisting of:

a. Binary split (It splits only in two direction)
b. Multiway split (It splits in multiple direction)

Thus, there are many challenges constructing a tree. They are mainly concerned with how to determine the best split, how to choose if we should go for a multiway or a single split. This is something that we will address in the next section when we will talk about our own implementation of the tree.

DECISION TREE ALGORITHM IMPLEMENTATION

1. Our implementation is based on HUNT's Algorithm.
2. In our implementation, we have a class ExecuteDecisionTree which contains all what it takes to create a tree programmatically.
3. This class has several functions, which are used to create a tree, do prediction, and then display a tree as well.
4. Internally we have used the strategy of binary split, since most of the data points provided to us were continuous variables, and also since the complexity of the tree was less in this case.
5. To do split and determine which the best split at any level is, we have used GINI Index.

6. Initially our program reads the file and divides the data into training and test sets.
7. This is achieved by the Class ReadFile.
8. This class returns training and test data samples after shuffling them.
9. Post this when we call the construct tree method, if takes the training data.
10. Then it determines the best split for each attribute based on the Gini index. In case if it is categorical variable it keeps one value on one side and the rest on the other to determine the best split.
11. Once the best split is determined, the other challenge is to figure out which attribute should be considered at this level out of all the current attributes.
12. For this, we again use Gini Index to determine the best information gain for any attribute. The attribute with the maximum information value is the one which is considered.
13. Then this becomes the parent node.
14. Then we split the records based on the nodes split value.
15. We get some records on the right and some on the left.
16. For records on the right we again calculate the best splits and the best attribute to be considered based on Gini Index.
17. Similarly for left as well we do the same thing.
18. Also, if at any stage all the attribute send the data in one direction it means there are replicated records and hence, the class will be same for all of them. So, the tree stops.
19. At any node if all the records in left or right are of the same class then for that direction as well there are no further expansions.
20. At the end the tree is a binary tree.

Structure of a Node:

```
package Part3;

public class Node {
    public int attributeIndex;
    public String splitValue;
    public String leftClassPredict;
    public String rightClassPredict;
    public Node leftChild;
    public Node rightChild;
}
```

NOTE: In our case when we find the training and the test data, we first import the file then shuffle the records and then partition it to get the training and test data. Since shuffle is random so if we will pull the training data and test data from file multiple times, every time model will be DIFFERENT since training data will be different.

HANDLING CONTINUOUS AND CATEGORICAL VARIABLE

In the dataset provided, there were both continuous and categorical variables. To handle them was a challenge in itself. It is worth remembering that we are doing binary splits. Hence, for categorical variables as well we had to do binary splits.

To identify if a variable was a categorical variable, we used java exception class, i.e. Double.parseDouble(value). If it threw exception that means there is a string. Once, we get a categorical variable we take one value at a time and identify Gini Indexes to see which value we need to choose for the best split.

This is how we handled categorical variable.

RESULT ANALYSIS

a. project3_dataset1.txt (90% training or 0.9)

Given below are the results of dataset 1. The algorithm gives different accuracies and measures every time because the training data comes after shuffle phase of the whole data set. Thus a different set of training data is chosen every time.

This is the result for a fairly good values that came up randomly. Also the tree structure is given as level order traversal. This is easy to comprehend. Since it is a binary tree, so there will be 2 child nodes and every stage has 2^n nodes.

The measure values A: Accuracy R: Recall P: Precision F: F Measure

In the data set 4 we will see how we can construct a tree given level order, since that data set is small.

```
Predicted LabelsList : [0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0
If records >= split value goes on right else left

|| 20:16.795:
|| 27:0.1563:|| 1:16.11:
|| 27:0.13579999999999998:|| 21:23.47:rightPredict1:|| 7:0.06626:leftPredict0:rightPredict1:|| 24:0.08798:leftPredict0:
|| 10:1.04755:rightPredict1:|| 1:20.785:rightPredict1:|| 4:0.12380000000000001:leftPredict0:rightPredict1:|| null|| null|| null|| nul
|| 10:0.6431:|| null|| 23:811.0999999999999:leftPredict0:|| null|| null|| null|| 1:20.674999999999997:leftPredict0:rightPredict1:|| n
|| 14:0.003309:|| 1:18.630000000000003:leftPredict1:rightPredict0:|| null|| 0:14.190000000000001:leftPredict1:rightPredict0:|| null||
|| 1:19.9:leftPredict0:rightPredict1:|| 21:33.349999999999994:leftPredict0:|| null|| null|| null|| null
|| null|| null|| null|| 21:33.56:leftPredict1:rightPredict0:
|| null|| null
A : 0.9824561403508771 | R : 0.9615384615384616 | F : 0.9803921568627451 | P : 1.0
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset1.txt", 0.9);
String[][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
ExecuteDecisionTree tree = new  ExecuteDecisionTree();
Node root = tree.constructTree(train);
ArrayList<String> label = tree.predictLabels(test, root);
System.out.println("Predicted LabelsList : " + label);
//tree.displayTree(root);
Measures m = new Measures(label, test);
m.displayMeasures();
```

Explanation of Results:

A high value of F measure means that the algorithm performs well in both precision and recall terms. Also, it is worth noting that the values of these parameters is low for 2nd dataset meaning that the first data set records better train the decision tree and hence, the data is more suitable for the tree.

b. **project3_dataset2.txt (90% training or 0.9)**

In this dataset there is a categorical variable. The values of which are present or absent. Also, the measure values for this dataset is lower compared with the dataset 1. It is for this reason that the dataset is not well suited for decision tree. Also, the node consists of the attribute under consideration by specifying its index value and the split value is also mentioned.

```
Predicted LabelsList : [0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1
If records >= split value goes on right else left

|| 8:50.0:
|| 8:31.0:|| 4:Present:
|| 0:182.0:rightPredict1:|| 5:68.5:|| 1:7.605:|| 1:0.8500000000000001:
|| 1:1.0550000000000002:|| null|| 2:3.705:|| 0:175.0:rightPredict0:|| 1:4.825:rightPredict0:|| 2:4.279999999999999:leftPredict1:|| 5:
|| 6:18.48:rightPredict0:|| 6:24.84:rightPredict0:|| 6:26.04:rightPredict0:|| 2:3.955:leftPredict1:|| 2:3.335:leftPredict0:rightPredi
|| 2:2.51:leftPredict1:rightPredict0:|| null|| 3:19.085:rightPredict1:|| null|| 0:151.0:rightPredict1:|| null|| null|| 6:19.985:leftP
|| null|| null|| 2:4.435:leftPredict0:rightPredict1:|| null|| 5:54.5:|| null|| null|| 2:6.275:|| 1:1.9:leftPredict1:rightPredict0:||
|| null|| null|| 0:142.0:leftPredict0:|| 4:Present:|| 7:13.68:|| 2:7.07:|| null|| null|| null|| 0:174.0:leftPredict1:rightPredict0:||
|| null|| 1:2.9000000000000004:leftPredict1:rightPredict0:|| 1:0.14:leftPredict1:|| 2:3.5949999999999998:leftPredict1:rightPredict0:|
|| null|| null|| null|| 3:25.65:leftPredict0:rightPredict1:|| null|| null|| 7:5.4:leftPredict0:|| 1:0.14:leftPredict0:rightPredict1:|
|| null|| null|| null|| 7:8.280000000000001:rightPredict0:|| null|| null|| null|| null|| null|| 0:134.0:|| null|| 6:26.0:leftPredict0
|| 0:147.0:rightPredict0:|| null|| 2:5.345000000000001:leftPredict1:rightPredict0:|| 0:167.0:leftPredict1:rightPredict0:|| null|| nul
|| 0:124.0:leftPredict0:rightPredict1:|| null|| null|| null|| null|| null|| null|| null
|| null|| null
A : 0.717391304347826 | R : 0.6666666666666666 | F : 0.6486486486486487 | P : 0.631578947368421
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset2.txt", 0.9);
String[][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
ExecuteDecisionTree tree = new  ExecuteDecisionTree();
Node root = tree.constructTree(train);
ArrayList<String> label = tree.predictLabels(test, root);
System.out.println("Predicted LabelsList : " + label);
tree.displayTree(root);
Measures m = new Measures(label, test);
m.displayMeasures();
```

Explanation of results:

Here we have used the decision tree only once, unlike random forest where there is a phenomenon of majority voting leading to correct classification. Also, it is worth remembering that decision tree is a weak classifier. Thus, the low precision value and F measure value are good to tell that the dataset prediction for this dataset is not that good.

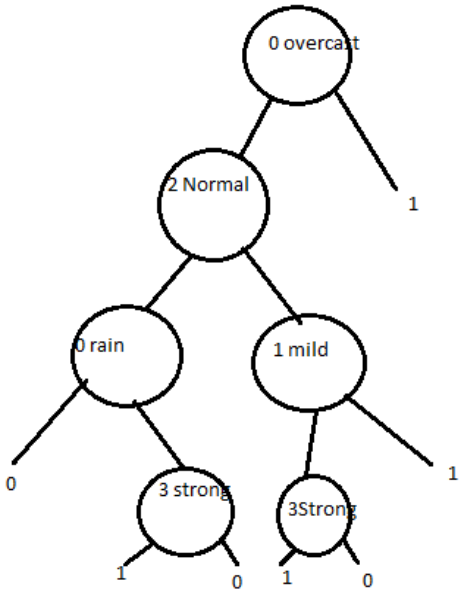
c. **project3_dataset4.txt (90% training or 0.9)**

Here, since the size of dataset is small, so after taking 90% training only one record was left for prediction. Since the prediction from the learnt tree is correct so all the measures are 100%.

Let us also see how to construct a tree from level order traversal. Below are some of the steps.

a. The first is the root node. 0 indicates the attribute under consideration. Overcast indicates the split value. Right predict 1 indicates that there is no right child and if someone goes to right side its class is 1. If records are equal to split value then they go to right side.
b. Now in the second row, since it is a binary tree so there can only be 2 nodes. Since from top the right child is null as explained so the second row node is a left child. The attribute value is 0 and split value is normal. Also, since there is no right predict and left predict hence, it means that it has children on both sides.
c. When we come to third row, from above we know that both nodes are children of the above node.
d. Here the first node has attribute 0, and split value rain. Also left Predict 0 means that there is no left child. And the class of record going left is 0. Similarly for second node the right predict 1 means there is no right child and the record going right is class 1.
e. Moving to the next stage since, above two node have left and right child null respectively so the remaining two are right and left child of the above two nodes.

Hence, this is how we can interpret the printed tree. The understood architecture from below is as below:




```
Predicted LabelsList : [1]
If records >= split value goes on right else left

|| 0:overcast:rightPredict1:
|| 2:normal:|| null
|| 0:rain:leftPredict0:|| 1:mild:rightPredict1:
|| null|| 3:strong:leftPredict1:rightPredict0:|| 3:strong:leftPredict1:rightPredict0:|| null
|| null|| null|| null|| null
A : 1.0 | R : 1.0 | F : 1.0 | P : 1.0
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset4.txt", 0.9);
String[][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
ExecuteDecisionTree tree = new  ExecuteDecisionTree();
Node root = tree.constructTree(train);
ArrayList<String> label = tree.predictLabels(test, root);
System.out.println("Predicted LabelsList : " + label);
tree.displayTree(root);
Measures m = new Measures(label, test);
m.displayMeasures();
```

Explanation:

From the above measures it is clear that when the prediction is correct, all the values are 1 or 100%. The tree structure is also presented above, which shows that it is easy to comprehend the tree from level order traversal.

PROS AND CONS

a. Decision trees are easy to understand.
b. They are very good in selecting features important to classify the data.
c. Complex data can be broken into simple trees for decision making.
d. Decision trees, while providing easy to view illustrations, can also be unwieldy. Even data that is perfectly divided into classes and uses only simple threshold tests may require a large decision tree
e. Among the major decision tree disadvantages are its complexity. Decision trees are easy to use compared to other decision-making models, but preparing decision trees, especially large ones with many branches, are complex and time-consuming affairs.
f. The reliability of the information in the decision tree depends on feeding the precise internal and external information at the onset. Even a small change in input data can at times, cause large changes in the tree.

K-FOLD CROSS VALIDATION

The K fold cross validation is done to check if the performance of decision tree remains uniform through the set of data. Also, since there can be many decision trees for a given data set hence, K fold cross validation helps us to know what is the average value of precision for a tree constructed on this particular set.

This helps us to understand if tree is suitable for a particular dataset. The strategy for K fold validation is simple. A dataset that became a training dataset in the first place now becomes the test data set. K fold means that at any time K% of the test records will be taken from the total data set and the remaining will be training set.

In our code, we took K% of records initially. Then we chose K% of records from the earlier used training set and merged the earlier used test set into the training set. This process keeps happening for K times. At the end we sort of get an average value of various measures signifying how the tree will perform on an average on a dataset.

Results of K-Fold Cross Validation on Decision Tree

a. *Project3_dataSet1.txt (K=15)*

```
A : 0.9529411764705882 | R : 0.9428571428571428 | F : 0.9428571428571428 | P : 0.9428571428571428
A : 0.9375 | R : 0.8787878787878788 | F : 0.90625 | P : 0.9354838709677419
A : 0.9270833333333334 | R : 0.9090909090909091 | F : 0.8955223880597015 | P : 0.8823529411764706
A : 0.9479166666666666 | R : 1.0 | F : 0.9295774647887324 | P : 0.868421052631579
A : 0.9791666666666666 | R : 0.9705882352941176 | F : 0.9705882352941176 | P : 0.9705882352941176
A : 0.9166666666666666 | R : 0.8837209302325582 | F : 0.9047619047619048 | P : 0.926829268292683
Avg Values :
A: 0.9435457516339869
R: 0.930840849377101
P: 0.9210887518699558
F: 0.9249261892935999
```

Explanation: For this dataset an average value of F measure over K = 15 gives a result suggesting that in general, tree performs well in all the cases on this dataset.

b. *Project3_dataSet2.txt (K=10)*

```
A : 0.6304347826086957 | R : 0.55 | F : 0.5641025641025641 | P : 0.5789473684210527
A : 0.717391304347826 | R : 0.6 | F : 0.5806451612903226 | P : 0.5625
A : 0.717391304347826 | R : 0.5 | F : 0.5185185185185185 | P : 0.5384615384615384
A : 0.717391304347826 | R : 0.5 | F : 0.5806451612903226 | P : 0.6923076923076923
A : 0.5 | R : 0.4 | F : 0.34285714285714286 | P : 0.3
A : 0.6521739130434783 | R : 0.4666666666666667 | F : 0.4666666666666667 | P : 0.4666666666666667
A : 0.6086956521739131 | R : 0.35714285714285715 | F : 0.35714285714285715 | P : 0.35714285714285715
A : 0.6739130434782609 | R : 0.5625 | F : 0.5454545454545454 | P : 0.5294117647058824
A : 0.6739130434782609 | R : 0.6 | F : 0.5454545454545454 | P : 0.5
A : 0.7608695652173914 | R : 0.7058823529411765 | F : 0.6857142857142857 | P : 0.6666666666666666
Avg Values :
A: 0.6652173913043479
R: 0.52421918767507
P: 0.5192104554372357
F: 0.5187201448491772
```

Explanation: Here the values are not as good as in the previous data set. For some selections of training set, the accuracy is better than the rest suggesting if training data is chosen from among those pockets it is likely that tree will give a better performance. But the average value of F measure tells us the general behavior of the classifier over this dataset.

c. *Project3_dataSet4.txt*

```
A : 0.6666666666666666 | R : 0.6666666666666666 | F : 0.8 | P : 1.0
A : 0.3333333333333333 | R : 1.0 | F : 0.5 | P : 0.3333333333333333
A : 0.6666666666666666 | R : 0.6666666666666666 | F : 0.8 | P : 1.0
A : 1.0 | R : 1.0 | F : 1.0 | P : 1.0
Avg Values :
A: 0.6666666666666666
R: 0.8333333333333333
P: 0.8333333333333333
F: 0.775
```

RANDOM FOREST

RANDOM FOREST ALGORITHM IMPLEMENTATION

Random forests are very useful in trees as they help to make predictions based upon majority voting and hence improve the performance of the tree. It is an ensemble learning approach. Here we bootstrap some part of data from the total dataset. Then we replace and replicate some data points in the selected data points. Similarly we choose several bootstrapped data points from the total set and hence, generate K- trees called a forest. The test data is predicted against each tree and finally the output is the majority say called ***Bagging***.

Implementation:

At the base we still use Decision Tree implementation. Some technical insights for Forest are as follows:

1. Each time we select number of trees as square root of size of training samples say K.
2. Post this we divide the training rows among these K number of trees where each training set also has some random replication of its records.
3. Once many training sets made and trees are formed we do prediction for a test set.
4. Based on K labels determined by K trees we take what the majority say is.

Result Analysis

a. project3_dataset1.txt

```
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.9210526315789473 | R : 0.8157894736842105 | F : 0.8732394366197183 | P : 0.9393939393939394
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset1.txt", 0.8);
String [][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
RandomForest forest = new RandomForest();
ArrayList<String> labels = forest.getLabelsForRandomForest(train, test);
Measures m = new  Measures(labels, test);
m.displayMeasures();
```

Analysis: The Forest on this dataset gives a good result. The measures signify that the bagging has worked well for predicting the labels of the test data. An F Measure of 87% suggests that both precision and recall are good. Hence, the tree is well suited for this data set.

b. project3_dataset2.txt

```
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.6847826086956522 | R : 0.41935483870967744 | F : 0.4727272727272727 | P : 0.5416666666666666
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset2.txt", 0.8);
String [][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
RandomForest forest = new RandomForest();
ArrayList<String> labels = forest.getLabelsForRandomForest(train, test);
Measures m = new  Measures(labels, test);
m.displayMeasures();
```

Analysis: For this dataset, forest yield a better accuracy and F measure than a single decision tree. The soul reason for this being the Bagging which helps in determining the result through majority voting.

c. project3_dataset4.txt

```
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.5 | R : 0.666666666666666 | F : 0.666666666666666 | P : 0.666666666666666
```

```
ReadFile rf = new ReadFile("B:/UB_CS/DataMining/Project3/Data/project3_dataset4.txt", 0.7);
String [][] train = rf.getTrainingData();
String [][] test = rf.getTestData();
RandomForest forest = new RandomForest();
ArrayList<String> labels = forest.getLabelsForRandomForest(train, test);
Measures m = new  Measures(labels, test);
m.displayMeasures();
```

Analysis: This is a small data set. Hence, the trees predicted for this dataset from forests are all small. Therefore the effect of majority voting may undermine the one good trees result. Since, we can have the multiple trees for same situation hence, it is hard to tell which performs well on which scenario. But forests give us a chance to incorporate maximum possibilities and hence, avoid erroneous predictions.

Pros and cons

1. It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
2. It runs efficiently on large databases.
3. It can handle thousands of input variables without variable deletion.
4. It gives estimates of what variables are important in the classification.

5. It generates an internal unbiased estimate of the generalization error as the forest building progresses.
6. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

K-Fold Cross Validation

In order to further enhance the performance of the trees and their accuracy while predicting, we can combine the two approaches of cross validation K folding and Forest. Now, in our implementation we calculate K fold, and hence pass training set and test set for K possibilities. For each possibility the Forest draws trees equal to the square root of training length. Hence, bagging and helps to predict labels for each iteration of K folding.

a. *Project3_dataSet1.txt*

Analysis

Here, for K fold multiple iterations the result of random forests is shown and along with this an average value of all the measures is calculated. The result signifies good algorithm performance with increased certainty of correct prediction.

```
Completed Bagging
A : 0.9166666666666666 | R : 0.8611111111111112 | F : 0.8857142857142857 | P : 0.9117647058823529
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.9375 | R : 0.8648648648648649 | F : 0.9142857142857143 | P : 0.9696969696969697
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.9166666666666666 | R : 0.8108108108108109 | F : 0.8823529411764706 | P : 0.967741935483871
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.9479166666666666 | R : 0.9655172413793104 | F : 0.9180327868852459 | P : 0.875
Avg Values :
A: 0.9331290849673203
R: 0.8899673380276828
P: 0.9307967556900195
F: 0.9082130221875012
```

b. *Project3_dataSet2.txt*

As seen from the above cases that data set 2 does not give good accuracy with tree. The cross validation on random forest proofs this even further. The reason being the fact that, it is a string predictive technique. The value of F measures are low suggesting a generic performance over many iterations is not that good.

```
Completed Bagging
A : 0.5869565217391305 | R : 0.47058823529411764 | F : 0.45714285714285713 | P : 0.4444444444444444
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.717391304347826 | R : 0.5294117647058824 | F : 0.5806451612903226 | P : 0.6428571428571429
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.6739130434782609 | R : 0.46153846153846156 | F : 0.4444444444444444 | P : 0.42857142857142855
Got the first Training Set
Got all training partitions
Started Random Forest
Completed Random Forest
Started Bagging
Completed Bagging
A : 0.7391304347826086 | R : 0.5909090909090909 | F : 0.6842105263157895 | P : 0.8125
Avg Values :
A: 0.691304347826087
R: 0.5028638028638028
P: 0.5598278548936444
F: 0.5230731119363897
```

Boosting

Algorithm Description –

We implemented Ada Boost algorithm also known as Adaptive Boost. The algorithm focuses on a simple strategy to bring in more and more weak classifiers, till your final misclassification rate for training data becomes arbitrarily small. Thus, boosting is used to figure out 2 things

1. It helps you choose the training set for each new classifier that you train based on the results of the previous classifier.

2. It determines how much weight should be given to each classifier's proposed answer when combining the results.

Algorithm Implementation –

We implemented algorithm in Java programming language. To perform boosting we require the training dataset. For this we have read the file and stored the training data in a 2D matrix. Below is the pseudo code of the algorithm-

Pseudo Code –

1) Read the data file and store the training dataset in a 2D matrix.
2) Iterate 10 times

    1) For the first iteration initialize the weight vector with initial weights (1/N) where, N is the number of training samples. Store the weight vector in a weight map with key as iteration number.
    2) Create a decision tree of the training data and get the predicted labels. Store the predicted labels on the training data in a predicted label map with key as iteration number. Also store the decision tree root node in a node array.
    3) Calculate the error by the below formula and store the error value in an error map with key as iteration number. Here Wj is the weight of the jth sample in previous iteration.

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

    4) Calculate alpha value by the below formula and store the alpha value in an alpha map with key as iteration number. Where, ei is the error of the ith iteration.

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

5) Calculate the new weights of the samples in training data by the below formula and store the new weights in the weights map with key as iteration number. Where, Wj is the weight of the jth sample in previous iteration, Yj is the actual label of the jth sample, C(Xj) is the predicted label of the jth sample.

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp\left(-\alpha_i y_j C_i(x_j)\right)}{Z^{(i)}}$$

6) From the second iteration onwards, generate a bootstrap sample from the original training data based on the weights. To do this first we get the weight array from the weight map. Then, we calculate the average of the weight array to find the average value of the weight. To find the bootstrap sample we get all the samples from the original training set with weights greater than the average weight. This way we get all the samples with were misclassified in the previous iteration.
7) We then generate a decision tree of the bootstrap sample and predict the labels of the training set from this decision tree.
8) Then we calculate the error value, alpha and update the weights.

3) Predict the labels of the test data set. For this we calculate the maximum alpha value from the alpha map. Iteration number of the maximum alpha value (key of alpha map) gives the root node of the decision tree from the node array. This decision tree is used to predict the labels on the test dataset. We used this approach because it is we observed the after reaching the maximum value of alpha the weights converged. Thus, predicting the labels using this approach is much easier using the combiner approach would also have served the same purpose.
4) Calculate the measure values for the predicted labels on the test dataset.

Experiment Results

1) Dataset – project3_dataset1.txt

Accuracy	0.84375
Precision	0.75
Recall	0.8617021276595744
F-Measure	0.801980198019802

```
Round: 1, Error: 0.95703125
Round: 2, Error: 0.8863636363636404
Round: 3, Error: 0.5000000000000002
Round: 4, Error: 0.4999999999999774
Round: 5, Error: 0.5000000000000131
Round: 6, Error: 0.49999999999998723
Round: 7, Error: 0.5000000000000127
Round: 8, Error: 0.4999999999999874
Round: 9, Error: 0.5000000000000133
Round: 10, Error: 0.4999999999999868
Round: 1, Aplha: -1.5516814688731773
Round: 2, Aplha: -1.0270618668477929
Round: 3, Aplha: -4.440892098500628E-16
Round: 4, Aplha: 4.5297099404704336E-14
Round: 5, Aplha: -2.6201263381154382E-14
Round: 6, Aplha: 2.5535129566377947E-14
Round: 7, Aplha: -2.531308496145421E-14
Round: 8, Aplha: 2.531308496145293E-14
Round: 9, Aplha: -2.6645352591004467E-14
Round: 10, Aplha: 2.6423307986078028E-14
Round: 1, Weight: [0.001953125, 0.001953125, 0.001953125, 0.001953125, 0.001953125, 0.001953125,
Round: 2, Weight: [0.009217576244614629, 0.009217576244614629, 0.009217576244614629, 0.009217576
Round: 3, Weight: [0.06347381838076976, 0.06347381838076976, 0.008137669023175288, 0.00813766902
Round: 4, Weight: [0.10000000000000338, 0.10000000000000338, 0.012820512820512735, 0.01282051282
Round: 5, Weight: [0.09999999999999615, 0.09999999999999615, 0.01282051282051297, 0.012820512820
Round: 6, Weight: [0.09999999999999896, 0.09999999999999896, 0.012820512820512659, 0.01282051282
Round: 7, Weight: [0.09999999999999608, 0.09999999999999608, 0.012820512820512943, 0.01282051282
Round: 8, Weight: [0.09999999999999885, 0.09999999999999885, 0.01282051282051265, 0.012820512820
Round: 9, Weight: [0.09999999999999606, 0.09999999999999606, 0.01282051282051294, 0.012820512820
Round: 10, Weight: [0.0999999999999991, 0.0999999999999991, 0.012820512820512647, 0.012820512820
Round: 11, Weight: [0.09999999999999609, 0.09999999999999609, 0.012820512820512936, 0.0128205128
Index of Max alpha4
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1,
A : 0.84375 | R : 0.8617021276595744 | F : 0.801980198019802 | P : 0.75
```

2) Dataset – project3_dataset2.txt

Accuracy	0.6081730769230769
Precision	0.45893719806763283
Recall	0.6506849315068494
F-Measure	0.5382436260623229

```
Round: 1, Error: 0.7860576923076924
Round: 2, Error: 0.8707865168539307
Round: 3, Error: 0.5000000000000132
Round: 4, Error: 0.4999999999999906
Round: 5, Error: 0.4999999999999996
Round: 6, Error: 0.49999999999999983
Round: 7, Error: 0.5
Round: 8, Error: 0.49999999999999983
Round: 9, Error: 0.5000000000000004
Round: 10, Error: 0.4999999999999995
Round: 1, Aplha: -0.6506619005825572
Round: 2, Aplha: -0.9539654504950401
Round: 3, Aplha: -2.6423307986079423E-14
Round: 4, Aplha: 1.8873791418627305E-14
Round: 5, Aplha: 8.881784197001244E-16
Round: 6, Aplha: 4.440892098500624E-16
Round: 7, Aplha: 0.0
Round: 8, Aplha: 4.440892098500624E-16
Round: 9, Aplha: -8.88178419700126E-16
Round: 10, Aplha: 1.1102230246251554E-15
Round: 1, Weight: [0.002403846153846154, 0.002403846153846154, 0.002403846153846154, 0.002403846153846154, 0.0
Round: 2, Weight: [0.004607714294010122, 0.004607714294010122, 0.004607714294010122, 0.004607714294010122, 0.0
Round: 3, Weight: [0.014584177077213482, 0.014584177077213482, 0.014584177077213482, 0.014584177077213482, 0.0
Round: 4, Weight: [0.021739130434783194, 0.021739130434783194, 0.021739130434783194, 0.021739130434783194, 0.0
Round: 5, Weight: [0.021739130434782646, 0.021739130434782646, 0.021739130434782646, 0.021739130434782646, 0.0
Round: 6, Weight: [0.021739130434782636, 0.021739130434782636, 0.021739130434782636, 0.021739130434782636, 0.0
Round: 7, Weight: [0.02173913043478264, 0.02173913043478264, 0.02173913043478264, 0.02173913043478264, 0.0217
Round: 8, Weight: [0.02173913043478265, 0.02173913043478265, 0.02173913043478265, 0.02173913043478265, 0.0217
Round: 9, Weight: [0.021739130434782643, 0.021739130434782643, 0.021739130434782643, 0.021739130434782643, 0.0
Round: 10, Weight: [0.021739130434782674, 0.021739130434782674, 0.021739130434782674, 0.021739130434782674, 0.
Round: 11, Weight: [0.02173913043478264, 0.02173913043478264, 0.02173913043478264, 0.02173913043478264, 0.0217
Index of Max alpha4
[1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1
A : 0.6081730769230769 | R : 0.6506849315068494 | F : 0.5382436260623229 | P : 0.45893719806763283
```

Result Analysis

In boosting, an interesting thing is the relationship between Error and alpha value. As the error increases, alpha value decreases and vice-versa. This interprets that the classifier weight grows exponentially as error approaches zero. Thus, better classifiers are given exponentially more weights in terms of importance. This is what we observed in both the datasets. As the values of alpha reaches maximum, the error gets minimized and we get the best classifier at this particular iteration of boosting. Also, at this iteration of boosting, the weights of the data points becomes stagnant and doesn't change much beyond this stable point. Thus, we performed the prediction of the test dataset using this best classifier.

Pros –

1) Very simple to implement.
2) Unlike SVM, boosting can achieve similar classification results with much less tweaking of parameters and settings. Only parameter is the number of iterations.
3) Provably effective in finding rough rules and hypothesis that are better than guessing.
4) Can be combined with any classifier like Decision Tree, Naïve Bayes, Neural Network etc.

Cons –

1) Can be sensitive to noise data and outliers.
2) Less susceptible to overfitting problem and good for generalization.
3) Gives a sub optimal solution.