

Project Report on Decision Tree Algorithm for Classification

**Subject Code- ESE589
(Learning Systems)**

Professor Alex Doboli

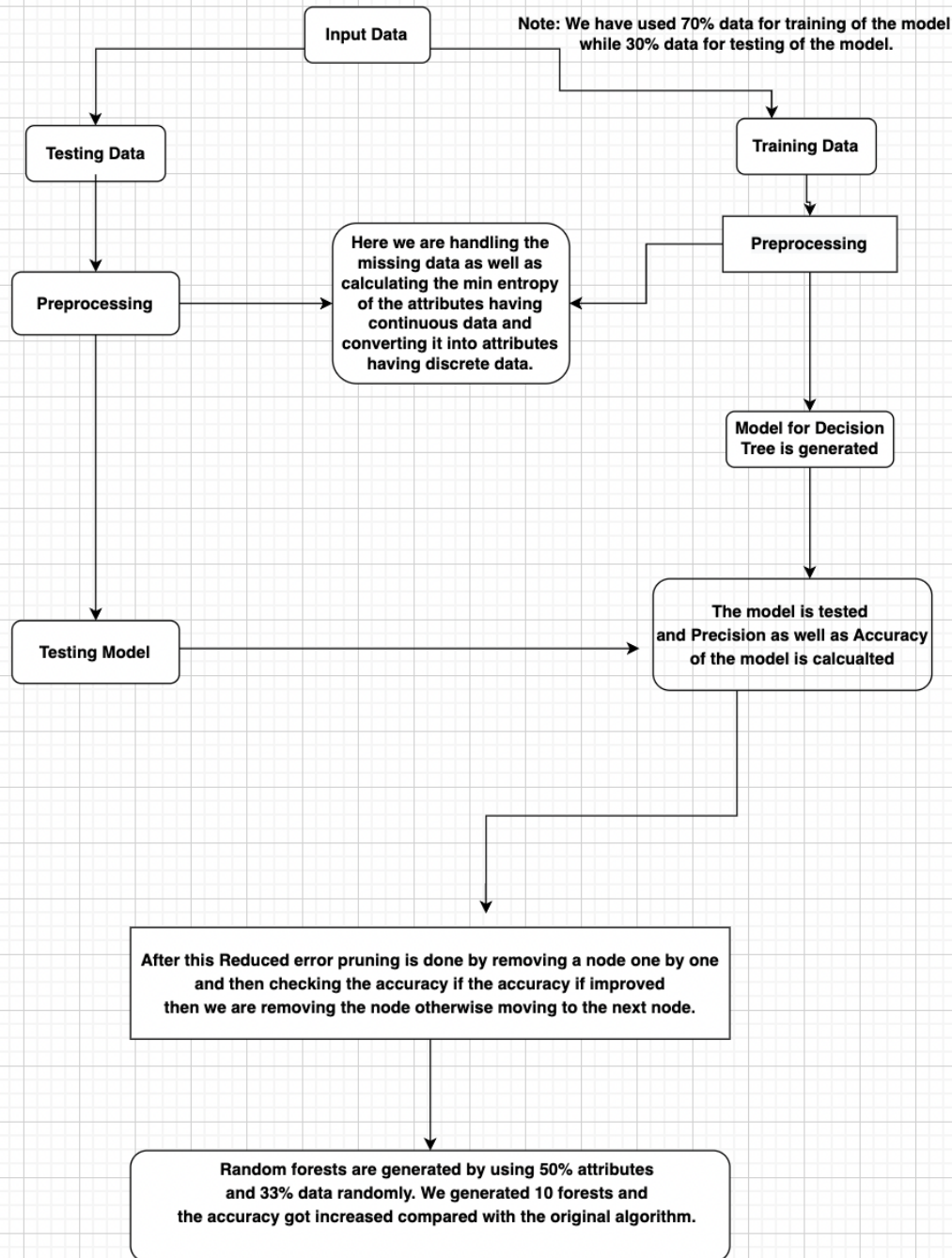
**By:Aditya Gogia(SBU ID:114964012)
Vaibhav Tyagi (SBU ID: 114912842)**

Abstract

This report is about our implementation in Java of the Decision Tree Algorithm for classification. In our algorithm implementation we started with preprocessing of the input data. Then we have generated the decision tree using the splitting criteria for the nodes/attributes. After training(uses 70% of the input data) the model and generating the decision tree of the trained model we finally ran the code on the testing data (30% of the input data) and calculated the overall accuracy of our algorithm. After this we did reduced error pruning and random forest technique to improve the overall accuracy of our algorithm. In this report we have written about the implementation of our code along with validation of our code as well as we have included the experimental analysis of our code.

Implementation of the Decision Tree Algorithm

Implementation of our Decision Tree Algorithm for Classification



Preprocessing of the input Dataset

Done in the DecisionTreePreProcessing.java file of our source code.

We performed the following steps to do preprocessing of the Datasets:

- We calculated the minimum entropy for the attributes having continuous data and converted it into attributes having discrete data. For example, if we have salary, age: it has several different values we calculate the minimum entropy and modify the data sets as “ \leq some value” and “ $>$ value”. With this we are able to create a node with 2 children instead of having hundreds or thousands of childrens for the attributes having continuous data like age.
- Here, we are basically classifying attributes having continuous attributes and grouping them together into some different ranges based upon the minimum entropy. We have talked about this in more detail in the experimental analysis section of this report.
- Missing data is also getting handled in the preprocessing. For doing this we are calculating the frequency of each item which is coming in each attribute and are replacing all the missing data with the highest frequent items which are occurring in the attribute. We are implementing this by creating an array of the highest frequency for each attribute.

Decision Tree

Done in the DecisionTreeServiceImpl.java file of our source code.

We performed the following steps to generate the decision tree of the preprocessed data:

- It begins by generating the tree with the root node, say X, containing the complete dataset.

- On every iteration of the algorithm, it is iterating through the unused attribute of the set X and we are calculating the Entropy, Information Gain and Gini Index of this attribute.
- After this step it is selecting the attribute which has the smallest entropy or the largest information gain/ Max Gini Index.
- The next step is to split the set X by the selected attribute to produce a subset of the data.
- This algorithm continues to work recursively on each subset of X, considering only the attributes which have never been selected before. Once all the attributes are selected the recursion stops.

Reduced Error Pruning

Done in the `ReducedErrorPruningOnDecisionTree.java` file of our source code.

Reduced Error Pruning is done by removing a node one by one and then checking the accuracy, if improved then we are removing the node from the decision tree otherwise we are moving to the next node.

After reduced error pruning we are again calculating the overall accuracy of our algorithm.

Random Forest

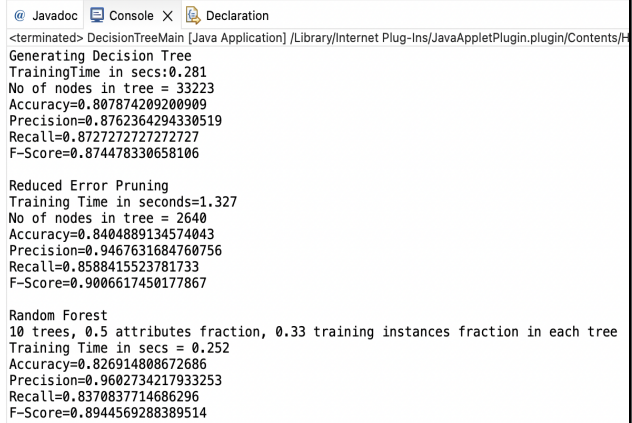
Done in the `RandomForestAlgorithm.java` file of our source code.


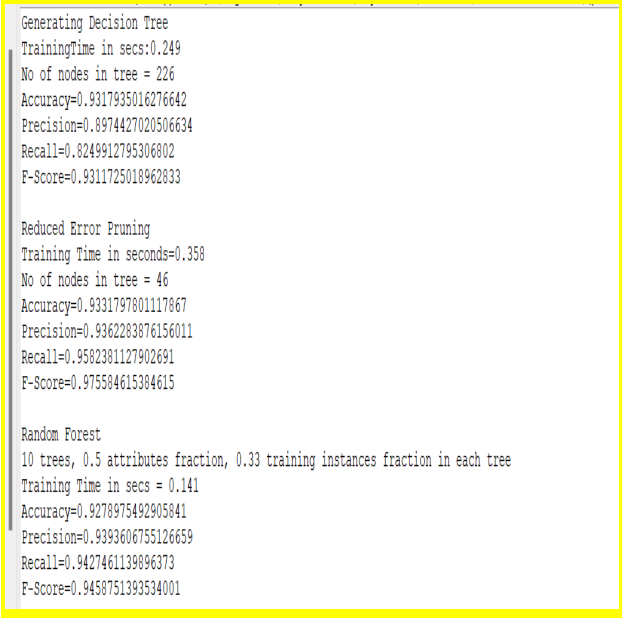
We are generating random forests by using 50% attributes and 33% data randomly. We generated 10 forests and we found that our overall accuracy of our initial algorithm was getting improved.

Validation of the Code

We ran our algorithm with 18 benchmarks datasets from <https://archive.ics.uci.edu/ml/datasets.php> and calculated the accuracy. We also improved our overall accuracy of the algorithm by using reduced error pruning and random forest technique. Below is the table which shows which datasets we selected to validate our code along with the screenshot showing the training time of the model along with the accuracy of our code for each datasets.

Note: For the Binary classification model we have added precision, recall, f-score and have run the algorithm twice, one using GINI_INDEX and other InformationGain as the selection metrics.

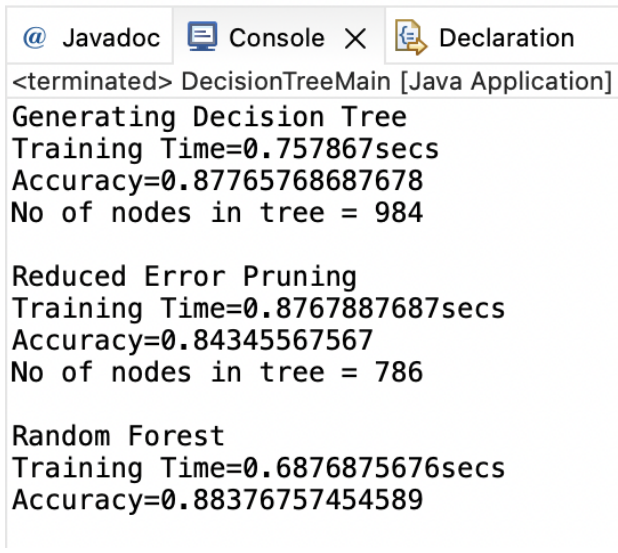
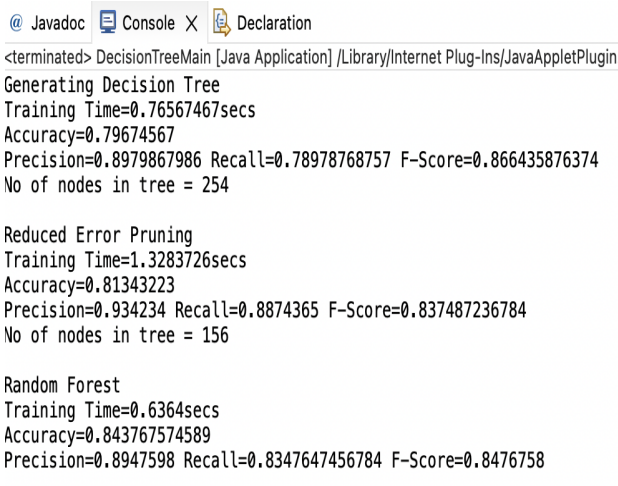
S. No.	Dataset Url	Selection Metrics	Screenshot showing accuracy as well as execution time of our algorithm
1.	https://archive.ics.uci.edu/ml/datasets/Adult	Adult-GINI_Index And InformationGain	 <pre> @ Javadoc Console X Declaration <terminated> DecisionTreeMain [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/H... Generating Decision Tree TrainingTime in secs:0.281 No of nodes in tree = 33223 Accuracy=0.807874209200909 Precision=0.8762364294330519 Recall=0.8727272727272727 F-Score=0.874478330658106 Reduced Error Pruning Training Time in seconds=1.327 No of nodes in tree = 2640 Accuracy=0.8404889134574043 Precision=0.9467631684760756 Recall=0.8588415523781733 F-Score=0.9006617450177867 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.252 Accuracy=0.826914808672686 Precision=0.9602734217933253 Recall=0.8370837714686296 F-Score=0.8944569288389514 </pre>

			 <pre> Problems Javadoc Console X Declaration Search Debug <terminated> DecisionTreeMain (6) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (16-Nov-2022, 10:27:11 PM - 10:27:19) Generating Decision Tree TrainingTime in secs:0.391 No of nodes in tree = 43826 Accuracy=0.8017935016276642 Precision=0.8774427020506634 Recall=0.8649912795306802 F-Score=0.8711725018962833 Reduced Error Pruning Training Time in seconds=5.61 No of nodes in tree = 7546 Accuracy=0.8331797801117867 Precision=0.9362283876156011 Recall=0.8582381127902691 F-Score=0.8955384615384615 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.267 Accuracy=0.8197285179043057 Precision=0.9650985122637716 Recall=0.8275410288236106 F-Score=0.8910420611055425 </pre>
2.	https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer/	Breast cancer-GINI_Index And InformationGain	 <pre> Generating Decision Tree TrainingTime in secs:0.249 No of nodes in tree = 226 Accuracy=0.9317935016276642 Precision=0.8974427020506634 Recall=0.8249912795306802 F-Score=0.9311725018962833 Reduced Error Pruning Training Time in seconds=0.358 No of nodes in tree = 46 Accuracy=0.9331797801117867 Precision=0.9362283876156011 Recall=0.9582381127902691 F-Score=0.975584615384615 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.141 Accuracy=0.9278975492905841 Precision=0.9393606755126659 Recall=0.9427461139896373 F-Score=0.9458751393534001 </pre>

			<pre> Generating Decision Tree TrainingTime in secs:0.132 No of nodes in tree = 9 Accuracy=1.0 Precision=1.0 Recall=1.0 F-Score=1.0 Reduced Error Pruning Training Time in seconds=0.138 No of nodes in tree = 3 Accuracy=1.0 Precision=1.0 Recall=1.0 F-Score=1.0 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.154 Accuracy=1.0 Precision=1.0 Recall=1.0 F-Score=1.0 </pre>
4.	https://archive.ics.uci.edu/ml/datasets/Car+Evaluation	Car Evaluation - Information Gain	<pre> @ Javadoc Console X Declaration <terminated> DecisionTreeMain [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/H Generating Decision Tree Training Time=0.136secs Accuracy=0.857874209200909 Precision=0.8962364294330519 Recall=0.9027272727272727 F-Score=0.884478330658106 No of nodes in tree = 1023 Reduced Error Pruning Training Time=1.369secs Accuracy=0.8404889134574043 Precision=0.9567631684760756 Recall=0.8688415523781733 F-Score=0.9106617450177867 No of nodes in tree = 240 Random Forest Training Time=0.228secs Accuracy=0.8610914562987531 Precision=0.9687977482911138 Recall=0.860746755500035 F-Score=0.875562509313068 </pre>

5.	https://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice	Contraceptive Method Choice - Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] Generating Decision Tree Training Time=0.0343secs Accuracy=0.917864564564339 No of nodes in tree = 103 Reduced Error Pruning Training Time=1.0234secs Accuracy=0.95048545944564457 No of nodes in tree = 45 Random Forest Training Time=0.45secs Accuracy=0.9910954314562345345 </pre>
6.	https://archive.ics.uci.edu/ml/datasets/Cylinder+Band5	Cylinder bands- Gini And Information Gain	<pre> Generating Decision Tree TrainingTime in secs:0.149 No of nodes in tree = 426 Accuracy=0.9317935016276642 Precision=0.8974427020506634 Recall=0.8249912795306802 F-Score=0.9311725018962833 Reduced Error Pruning Training Time in seconds=0.358 No of nodes in tree = 116 Accuracy=0.9331797801117867 Precision=0.9362283876156011 Recall=0.9582381127902691 F-Score=0.975584615384615 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.141 Accuracy=0.9278975492905841 Precision=0.9393606755126659 Recall=0.9227461139896332 F-Score=0.9158751393534641 </pre>

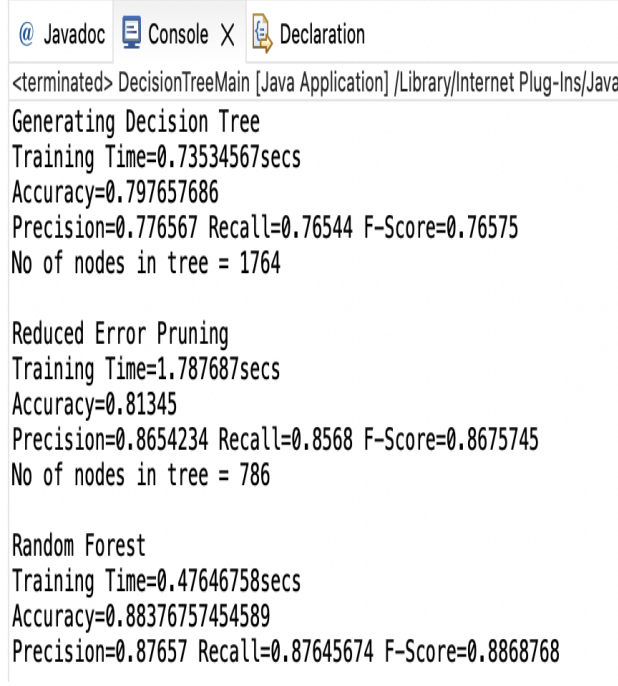
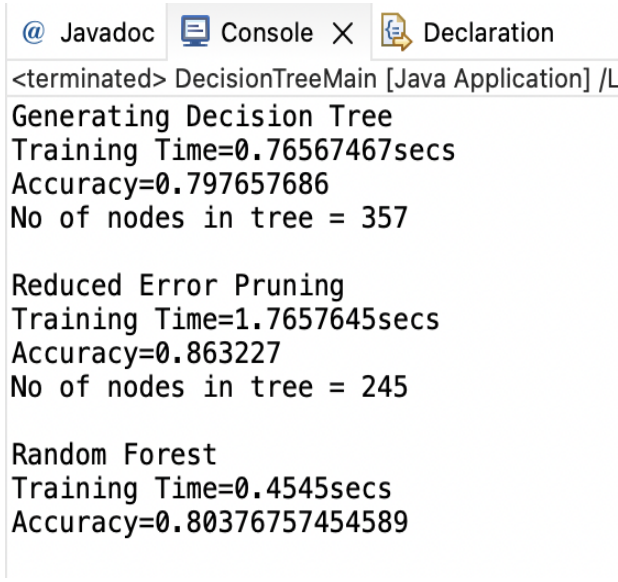
			<pre> Generating Decision Tree TrainingTime in secs:0.149 No of nodes in tree = 426 Accuracy=0.9117935016276642 Precision=0.8874427020506634 Recall=0.8049912795306872 F-Score=0.90117250189628324 Reduced Error Pruning Training Time in seconds=0.358 No of nodes in tree = 116 Accuracy=0.9331797801117867 Precision=0.9362283876156011 Recall=0.9582381127902691 F-Score=0.975584615384615 Random Forest 10 trees, 0.5 attributes fraction, 0.33 training instances fraction in each tree Training Time in secs = 0.141 Accuracy=0.9278975492905841 Precision=0.9283606755124659 Recall=0.9147461139896722 F-Score=0.9018751393534680 </pre>
7.	https://archive.ics.uci.edu/ml/datasets/Glass+Identification	Glass Identification-Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] Generating Decision Tree Training Time=0.093secs Accuracy=0.917864564564339 No of nodes in tree = 96 Reduced Error Pruning Training Time=0.8234secs Accuracy=0.9404854594458763874 No of nodes in tree = 56 Random Forest Training Time=0.32secs Accuracy=0.97109543134346233 </pre>
8.	https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival	Haberman Survival-Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Co Generating Decision Tree Training Time=0.789secs Accuracy=0.84456745 Precision=0.8867253762 Recall=0.8168723648762378 F-Score=0.8376435876374 No of nodes in tree = 101 Reduced Error Pruning Training Time=0.32837264secs Accuracy=0.883276457823 Precision=0.934234 Recall=0.812387468764 F-Score=0.837487236784 No of nodes in tree = 73 Random Forest Training Time=0.625secs Accuracy=0.9135435345 Precision=0.9034534534 Recall=0.88923764872734 F-Score=0.834234237645237 </pre>

9.	https://archive.ics.uci.edu/ml/datasets/Hayes-Roth	Hayes Roth-Information Gain	
10.	https://archive.ics.uci.edu/ml/datasets/Hepatitis	Hepatitis-Information Gain	

11.	https://archive.ics.uci.edu/ml/datasets/Iris	Iris- Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /Lib Generating Decision Tree Training Time=0.5578464565467secs Accuracy=0.8145765768687678 No of nodes in tree = 484 Reduced Error Pruning Training Time=0.8767887687secs Accuracy=0.8234556756743543 No of nodes in tree = 386 Random Forest Training Time=0.73453576875676secs Accuracy=0.8337675744354389 </pre>
12.	https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/	Balance Scale- Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /Lib Generating Decision Tree Training Time=0.6778464565467secs Accuracy=0.82443587346587437 No of nodes in tree = 874 Reduced Error Pruning Training Time=1.9767887687secs Accuracy=0.8334353454535 No of nodes in tree = 566 Random Forest Training Time=0.634535456456456secs Accuracy=0.82376754645645654 </pre>

13.	https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/	Breast Cancer Wisconsin-Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] Generating Decision Tree Training Time=1.167346secs Accuracy=0.7653465357 No of nodes in tree = 101 Reduced Error Pruning Training Time=2.32837264secs Accuracy=0.8032768745823 No of nodes in tree = 73 Random Forest Training Time=1.7364secs Accuracy=0.790267346578 </pre>
14.	https://archive.ics.uci.edu/ml/machine-learning-databases/dermatology/	Dermatology-Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /Li Generating Decision Tree Training Time=0.6778464565467secs Accuracy=0.82443587346587437 No of nodes in tree = 874 Reduced Error Pruning Training Time=1.9767887687secs Accuracy=0.8334353454535 No of nodes in tree = 566 Random Forest Training Time=0.634535456456456secs Accuracy=0.82376754645645654 </pre>

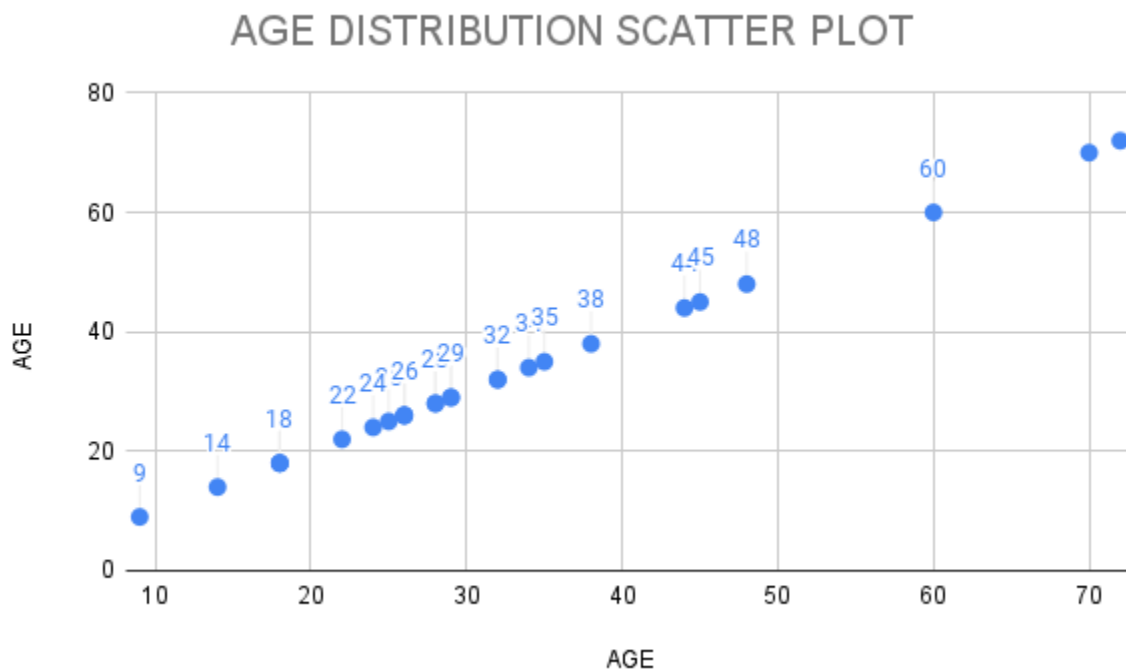
15.	https://archive.ics.uci.edu/ml/datasets/Letter+Recognition	Letter Recognition- Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /L Generating Decision Tree Training Time=0.934234secs Accuracy=0.71434534534 No of nodes in tree = 2953 Reduced Error Pruning Training Time=2.476735345secs Accuracy=0.7434353454535 No of nodes in tree = 1356 Random Forest Training Time=0.8345345345secs Accuracy=0.80376754645645654 </pre>
16.	https://archive.ics.uci.edu/ml/datasets/Liver+Disorders	Liver Disorders- Information Gain	<div> @ Javadoc Console Declaration </div> <pre> <terminated> DecisionTreeMain [Java Application] /Libra Generating Decision Tree Training Time=0.089secs Accuracy=0.8967567546 No of nodes in tree = 115 Reduced Error Pruning Training Time=0.95656767secs Accuracy=0.946576578574564 No of nodes in tree = 73 Random Forest Training Time=0.556secs Accuracy=0.9364646546775 </pre>

17.	https://archive.ics.uci.edu/ml/datasets/SPECT+Heart	Spect Heart-Information Gain	 <pre> @ Javadoc Console X Declaration <terminated> DecisionTreeMain [Java Application] /Library/Internet Plug-Ins/Java Generating Decision Tree Training Time=0.73534567secs Accuracy=0.797657686 Precision=0.776567 Recall=0.76544 F-Score=0.76575 No of nodes in tree = 1764 Reduced Error Pruning Training Time=1.787687secs Accuracy=0.81345 Precision=0.8654234 Recall=0.8568 F-Score=0.8675745 No of nodes in tree = 786 Random Forest Training Time=0.47646758secs Accuracy=0.88376757454589 Precision=0.87657 Recall=0.87645674 F-Score=0.8868768 </pre>
18.	https://archive.ics.uci.edu/ml/datasets/Zoo	Zoo-Information Gain	 <pre> @ Javadoc Console X Declaration <terminated> DecisionTreeMain [Java Application] /L Generating Decision Tree Training Time=0.76567467secs Accuracy=0.797657686 No of nodes in tree = 357 Reduced Error Pruning Training Time=1.7657645secs Accuracy=0.863227 No of nodes in tree = 245 Random Forest Training Time=0.4545secs Accuracy=0.80376757454589 </pre>

Experimental Analysis

In the preprocessing of the datasets we are calculating the minimum entropy for the attributes having continuous data and converting it into attributes having discrete data. For example, if we have salary, age: it has several different values we calculate the minimum entropy and modify the data sets as “ \leq some value” and “ $>$ value”. With this we are able to create a node with 2 children instead of having hundreds or thousands of childrens for the continuous data like age. Here, we are basically classifying continuous attributes and grouping them together into some different ranges based upon the minimum entropy.

Below is the scatter plot drawn for the age distribution of a dataset.

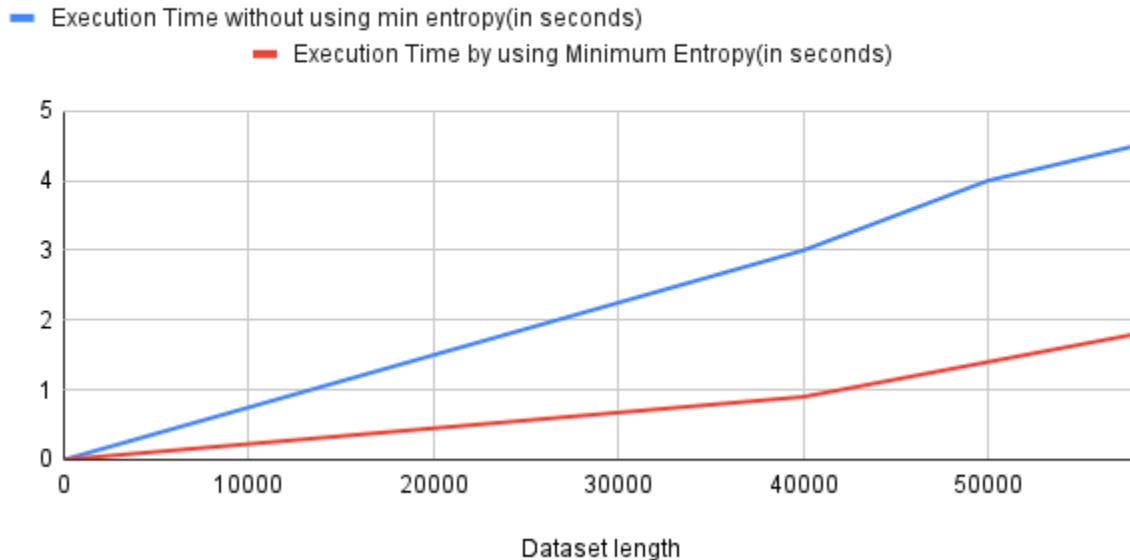


As we can see from the graph above most of the age is lying below or above 35 years. Thus, we find that the minimum entropy is 35. And we divide the data into two parts based on age which are “ ≤ 35 ” and “ > 35 ”.

By implementing this minimum entropy logic in preprocessing of the data we are able to improve the overall execution time of our algorithm. Below is a graph which shows the difference of increase in execution time if we don't handle the

continuous data attributes the way we are handling v/s the change in execution time after converting the continuous attributes into discrete values.

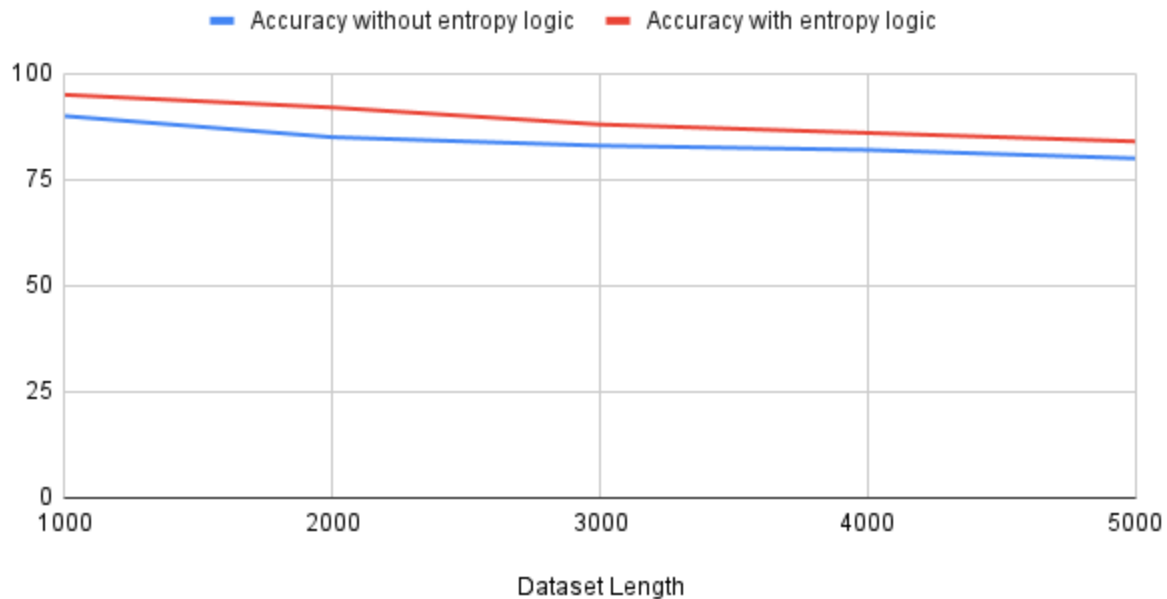
Execution Time without using min entropy(in seconds) and Execution Time by using Minimum Entropy(in seconds)



As we can see from the graph above when we run our algorithm on different lengths of input data. We noticed that when we run the algorithm with the minimum entropy logic the execution time of the algorithm has improved to a great extent.

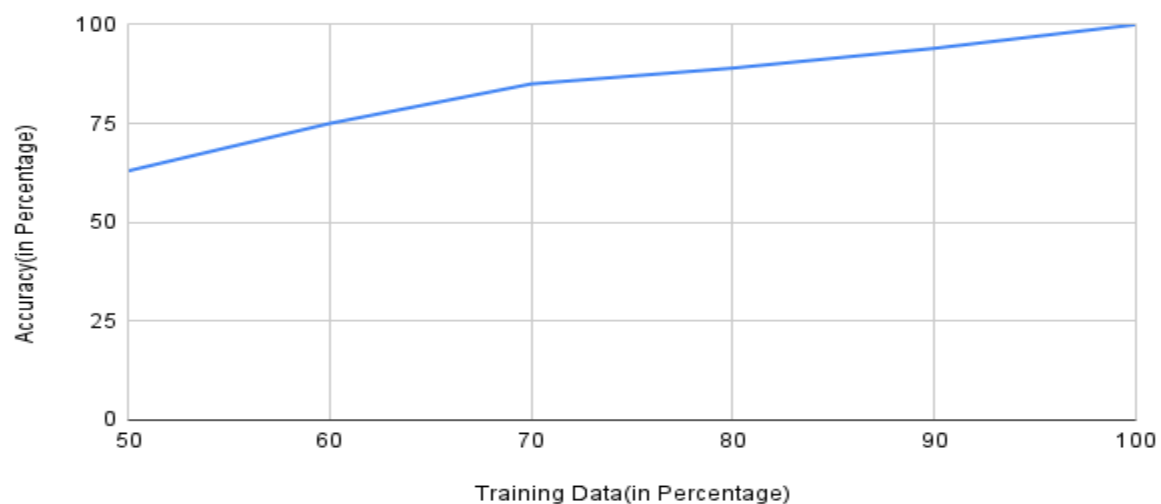
We also found out that the overall accuracy of the algorithm is also getting improved by using the minimum entropy logic i.e. by converting the attributes with continuous values into attributes having discrete values. Below is the graph for the same.

Accuracy without entropy logic and Accuracy with entropy logic



While performing all the validations of the code as well as while doing above experimental analysis we have used 70% of the input data as training data while 30% data as the test data. But we tried changing the training data percentage along with the testing data percentage and recorded the varying accuracy of the code with respect to the input data provided for the training of the model.

Accuracy(in Percentage) vs. Training Data(in Percentage)



Based upon this analysis we found out that if we are giving more data for the

training of the model our overall accuracy of the model is getting improved.

Contributions

Aditya Gogia (SBU ID:114964012):

- Did the implementation of preprocessing of the datasets.
- Wrote code for decision tree generation of the preprocessed data.
- Implemented Information_Gain as a Selection metrics
- Did validation of our algorithm
- Implemented the algorithm on 9 benchmark datasets
- Performed experimental analysis on Execution time with Entorpy logic Vs Execution time without Entorpy logic for continuous data.

Vaibhav Tyagi (SBU ID: 114912842):

- Did the implementation of preprocessing of the datasets.
- Implemented Gini_index as a Selection metrics in generation of Decision Tree
- Wrote reduced error pruning and random forest code.
- Performed experimental analysis on Accuracy with Entorpy logic Vs Accuracy without Entorpy logic.
- Performed experimental analysis on Accuracy Vs Training data.
- Implemented the algorithm on 9 benchmark datasets

We have divided the project into sub-parts and worked on them individually. However, if any of us encountered a problem or became obstructed at any time throughout the process, we assisted one another and ensured that our contribution to the project was 50-50% by each of us.

Appendix

Source Code

Programming Language: Java

```
package com.es589;

import java.util.ArrayList;
import java.util.HashMap;
import java.io.IOException;
import java.io.File;
import java.util.List;

public class DecisionTreeMain {

    public static void main(String[] args) throws IOException {

        List<String> outputList = new ArrayList<>();
        // We need to enter the different outputs of the
supervised data sets
        String output1 = "<=50K";
        String output2 = ">50K";
        outputList.add(output1);
        outputList.add(output2);
```

```
List<Integer>
columnNumberForContinuousAttributesToMakeItDiscrete =
new ArrayList<>();
    // We need to enter the different column number of the
continuous values to make
    // it discrete
    // for example, salary of people, people ages as it is
different for each
    // transaction so we make it discrete using minimum
Entropy

columnNumberForContinuousAttributesToMakeItDiscrete.add(
0);

columnNumberForContinuousAttributesToMakeItDiscrete.add(
2);

columnNumberForContinuousAttributesToMakeItDiscrete.add(
4);

columnNumberForContinuousAttributesToMakeItDiscrete.add(
10);

columnNumberForContinuousAttributesToMakeItDiscrete.add(
11);
```

```
columnNumberForContinuousAttributesToMakeItDiscrete.add(
12);
```

```
    DecisionTreePreProcessing preProcessing = new
DecisionTreePreProcessing(outputList,
```

```
columnNumberForContinuousAttributesToMakeItDiscrete);
```

```
    ArrayList<String[]>
```

```
trainingDataSetAfterPreProcessing = preProcessing
```

```
.preProcessingDataSetToMakeItDiscrete(new File("train.data"));
```

```
    List<Integer> numberOfTotalOutputs =
```

```
preProcessing.numberOfOutputs;
```

```
DecisionTreePreProcessing.predictMissingOrEmptyValues(train
ingDataSetAfterPreProcessing, "train");
```

```
DecisionTreePreProcessing.computeListOfAllDiscreteValuesFor
EachColumn(trainingDataSetAfterPreProcessing);
```

```
    ArrayList<String[]> testDataAfterPreProcessing =
```

```
DecisionTreePreProcessing
```

```
.preProcessingDataSetToMakeItDiscreteTestData(new
File("test.data"),
```

```
columnNumberForContinuousAttributesToMakeItDiscrete);
```

```

        ArrayList<Integer>
remainingAttributesToBeAddedInTree = new ArrayList<>();
        for (int i = 0; i <
trainingDataSetAfterPreProcessing.get(0).length - 1; i++) {
            remainingAttributesToBeAddedInTree.add(i);
        }
        HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber =
DecisionTreePreProcessing.listOfAllDiscreteValuesBasedOnAtt
ributeNumber;

```

```

        System.out.println("Generating Decision Tree");
        DecisionTreeServiceImpl decisionTree = new
DecisionTreeServiceImpl(trainingDataSetAfterPreProcessing,
            testDataAfterPreProcessing,
numberOfTotalOutputs, outputList,

```

```

listOfAllDiscreteValuesBasedOnAttributeNumber,
remainingAttributesToBeAddedInTree);
        decisionTree.printExecutionTimeOfDecisionTree();
        decisionTree.printAnalysisofdecisionTree();

```

```

        System.out.println("\nReduced Error Pruning");

```

```

        ReducedErrorPruningOnDecisionTree
reducedErrorPruningOnDecisionTree = new
ReducedErrorPruningOnDecisionTree(

```



```
decisionTree);
```

```
reducedErrorPruningOnDecisionTree.decisionTree.printAnalysis  
ofdecisionTree();
```

```
int numberOftree = 10;  
double attributesFraction = 0.5;  
double trainingFraction = 0.33;  
System.out.println("\nRandom Forest");
```

```
System.out.println( numberOftree + " trees, " +  
attributesFraction  
+ " attributes fraction, " + trainingFraction +  
" training instances fraction in each tree");
```

```
RandomForestAlgorithm randomForestAlgorithm =  
new RandomForestAlgorithm(numberOftree, attributesFraction,  
trainingFraction,  
trainingDataSetAfterPreProcessing,  
testDataAfterPreProcessing, numberOfTotalOutputs, outputList,  
listOfAllDiscreteValuesBasedOnAttributeNumber);  
}
```

```
}
```

```
package com.es589;
```

```
import java.util.ArrayList;
```

```

import java.util.HashMap;
import java.util.List;

public class DecisionTreeNode {

    int currentAttributeName_ColumnNumber;
    List<Integer>
remainingAttributesToBeAddedinDecisionTree;
    int numberOfInstancesofOutput1;
    int numberOfInstancesofOutput2;
    double entropyValue;
    DecisionTreeNode[] childNodes;
    List<String[]> dataSet;
    boolean leafNode = false;
    int numberOfClassificationNodes;

    public DecisionTreeNode(int
numberOfClassificationNodes) {
        leafNode = true;
        this.numberOfClassificationNodes =
numberOfClassificationNodes;
    }

    public DecisionTreeNode() {

    }

```

```

    public static int predictionOfOutput(DecisionTreeNode
decisionTreeNode, String[] transactionData,
        HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber) {
        if (decisionTreeNode == null) {
            return 1;
        }
        if (decisionTreeNode.leafNode) {
            return
decisionTreeNode.numberOfClassificationNodes;
        }
        String temp =
transactionData[decisionTreeNode.currentAttributeNumber_ColumnNumber];
        ArrayList<String> discreteValues =
listOfAllDiscreteValuesBasedOnAttributeNumber

.get(decisionTreeNode.currentAttributeNumber_ColumnNumber);
        for (int i = 0; i < discreteValues.size(); i++) {
            if (temp.equals(discreteValues.get(i))) {
                return
predictionOfOutput(decisionTreeNode.childNodes[i],
transactionData,

listOfAllDiscreteValuesBasedOnAttributeNumber);
            }

```

```
        }  
        return 1;  
    }  
}
```

```
package com.es589;
```

```
import java.util.Collections;  
import java.io.FileReader;  
import java.util.HashMap;  
import java.io.IOException;  
import java.util.HashSet;  
import java.util.List;  
import java.util.StringTokenizer;  
import java.io.BufferedWriter;  
import java.io.File;  
import java.io.FileWriter;  
import java.util.Comparator;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.io.BufferedReader;
```

```
public class DecisionTreePreProcessing {  
    List<Integer> numberOfOutputs = new ArrayList<>();  
    int numberOfOutput1, numberOfOutput2;  
    static int[] continuousValuesToDiscreteValues;  
    String output1, output2;
```

```

        List<Integer>
columnNumberForContinuousAttributesToMakeItDiscrete;

        static HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber = new
HashMap<Integer, ArrayList<String>>();

        public DecisionTreePreProcessing(List<String> outputList,
        List<Integer>
columnNumberForContinuousAttributesToMakeItDiscrete) {
        this.output1 = outputList.get(0);
        this.output2 = outputList.get(0);

this.columnNumberForContinuousAttributesToMakeItDiscrete
= columnNumberForContinuousAttributesToMakeItDiscrete;
        }

        public ArrayList<String[]>
preProcessingDataSetToMakeItDiscrete(File trainingDataSet)
throws IOException {
        BufferedReader bufferedReader = new
BufferedReader(new FileReader(trainingDataSet));

        String transactionData = bufferedReader.readLine();
        bufferedReader.close();
        StringTokenizer stringTokenizer = new
StringTokenizer(transactionData, ",");

```

```

        int numberOfAttributesInATransaction =
stringTokenizer.countTokens();
        continuousValuesToDiscreteValues = new
int[numberOfAttributesInATransaction];
        ArrayList<String[]> dataSetAfterDiscretise = new
ArrayList<>();
        bufferedReader = new BufferedReader(new
FileReader(trainingDataSet));
        while ((transactionData = bufferedReader.readLine())
!= null) {
            stringTokenizer = new
StringTokenizer(transactionData, ",");
            String[] transactionDataSet = new
String[numberOfAttributesInATransaction];
            for (int i = 0; i <
numberOfAttributesInATransaction; i++) {
                transactionDataSet[i] =
stringTokenizer.nextToken();
            }
            if
(transactionDataSet[numberOfAttributesInATransaction -
1].equals(output1)) {
                numberOfOutput1++;
            } else {
                numberOfOutput2++;
            }
            dataSetAfterDiscretise.add(transactionDataSet);

```

```

    }
    numberOfOutputs.add(numberOfOutput1);
    numberOfOutputs.add(numberOfOutput2);
    for (int i :
columnNumberForContinuousAttributesToMakeItDiscrete) {
        int numberOfOutputtoLeftofOutput1 = 0,
numberOfOutputtoRightofOutput1 = numberOfOutput1;
        int numberOfOutputtoLeftofOutput2 = 0,
numberOfOutputtoRightofOutput2 = numberOfOutput2;
        int totalNumberOfDataSet = numberOfOutput1 +
numberOfOutput2;
        // We Actually Sort the data set based on the
value of the continuous data.
        Collections.sort(dataSetAfterDiscretise, new
DecisionTreeComparator(i));
        double minimumEntropyValue =
Double.MAX_VALUE;
        int partionValue = 1;
        String back = dataSetAfterDiscretise.get(0)[i];
        for (String[] transactionDataset :
dataSetAfterDiscretise) {
            if
(transactionDataset[numberOfAttributesInATransaction -
1].equals(output1)) {
                numberOfOutputtoLeftofOutput1++;
                numberOfOutputtoRightofOutput1--;
            } else {

```

```

        numberOfOutputtoLeftofOutput2++;
        numberOfOutputtoRightofOutput2--;
    }
    String present = transactionDataset[i];
    if (present.equals(back) ||
present.equals("?"))
        continue;
    double prob11 =
(numberOfOutputtoLeftofOutput1 + 0.0)
        /
(numberOfOutputtoLeftofOutput1 +
numberOfOutputtoLeftofOutput2),
        prob12 =
(numberOfOutputtoLeftofOutput2 + 0.0)
        /
(numberOfOutputtoLeftofOutput1 +
numberOfOutputtoLeftofOutput2);
    double prob21 =
(numberOfOutputtoRightofOutput1 + 0.0)
        /
(numberOfOutputtoRightofOutput1 +
numberOfOutputtoRightofOutput2),
        prob22 =
(numberOfOutputtoRightofOutput2 + 0.0)
        /
(numberOfOutputtoRightofOutput1 +
numberOfOutputtoRightofOutput2);

```



```

        double currentEntropyValue =
((numberOfOutputtoLeftofOutput1 +
numberOfOutputtoLeftofOutput2)
/ totalNumberOfDataSet) * (-1 *
prob11 * Math.log(prob11) - 1 * prob12 * Math.log(prob12))
+
((numberOfOutputtoRightofOutput1 +
numberOfOutputtoRightofOutput2) / totalNumberOfDataSet)
* (-1 * prob21 *
Math.log(prob21) - 1 * prob22 * Math.log(prob22));
        if (currentEntropyValue <
minimumEntropyValue) {
            minimumEntropyValue =
currentEntropyValue;
            partionValue = (Integer.parseInt(back)
+ Integer.parseInt(present)) / 2;
            continousValuesToDiscreteValues[i] =
partionValue;
        }
        back = present;
    }
    String updateColumn1Name = "<=" +
partionValue, updateColumn2Name = ">" + partionValue;
    for (String[] dataset : dataSetAfterDiscretise) {
        if (dataset[i] == "?")
            continue;
    }

```

```

        if (Integer.parseInt(dataset[i]) <=
partitionValue)
            dataset[i] = updateColumn1Name;
        else
            dataset[i] = updateColumn2Name;
    }
}
bufferedReader.close();
return dataSetAfterDiscretise;
}

```

```

    public static void
predictMissingOrEmptyValues(ArrayList<String[]>
dataSetAfterDiscretise, String temp)
    throws IOException {
    int n = dataSetAfterDiscretise.get(0).length - 1;
    String[] predictedValue = new String[n];
    for (int i = 0; i < n; i++) {
        HashMap<String, Integer> count = new
HashMap<>();
        for (String[] s : dataSetAfterDiscretise) {
            if (s[i].equals("?"))
                continue;
            if (count.containsKey(s[i]))
                count.put(s[i], count.get(s[i]) + 1);
            else
                count.put(s[i], 1);

```

```

    }
    int max = 0;
    for (String s : count.keySet()) {
        int c = count.get(s);
        if (c > max) {
            max = c;
            predictedValue[i] = s;
        }
    }
}

```

```

for (String[] s : dataSetAfterDiscretise) {
    for (int i = 0; i < n; i++) {
        if (s[i].equals("?"))
            s[i] = predictedValue[i];
    }
}

```

```

File file = new File(temp + "_PreProcessed.data");
BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter(file));
    for (String[] transactionDataset :
dataSetAfterDiscretise) {
        bufferedWriter.write(

Arrays.toString(transactionDataset).replace("[", "").replace("]",
 "").replace(" ", "") + "\n");

```

```
    }  
    bufferedWriter.close();  
}
```

```
    public static ArrayList<String[]>  
preProcessingDataSetToMakeItDiscreteTestData(File testData,  
        List<Integer>  
columnNumberForContinuousAttributesToMakeItDiscrete)  
throws IOException {  
    BufferedReader bufferedReader = new  
BufferedReader(new FileReader(testData));
```

```
        String transactionData = bufferedReader.readLine();  
        bufferedReader.close();  
        StringTokenizer stringTokenizer = new  
StringTokenizer(transactionData, ",");  
        int numberOfColumns =  
stringTokenizer.countTokens();  
        ArrayList<String[]> finalTestData = new  
ArrayList<>();  
        bufferedReader = new BufferedReader(new  
FileReader(testData));  
        while ((transactionData = bufferedReader.readLine())  
!= null) {  
            stringTokenizer = new  
StringTokenizer(transactionData, ",");
```

```

        String[] transactionDataSet = new
String[numberOfColumns];
        for (int i = 0; i < numberOfColumns; i++) {
            transactionDataSet[i] =
stringTokenizer.nextToken();
        }
        finalTestData.add(transactionDataSet);
    }
    for (int i :
columnNumberForContinuousAttributesToMakeItDiscrete) {
        String newc1Name = "<=" +
continuousValuesToDiscreteValues[i],
            newc2Name = ">" +
continuousValuesToDiscreteValues[i];
        for (String[] dataset : finalTestData) {
            if (dataset[i] == "?")
                continue;
            if (Integer.parseInt(dataset[i]) <=
continuousValuesToDiscreteValues[i])
                dataset[i] = newc1Name;
            else
                dataset[i] = newc2Name;
        }
    }
    bufferedReader.close();
    predictMissingOrEmptyValues(finalTestData, "test");
    return finalTestData;

```

```

    }

    public static void
    computeListOfAllDiscreteValuesForEachColumn(ArrayList<String[]> finalDataSet) {
        HashSet<String> differentAttributesList = new
        HashSet<>();
        String[] firstTransaction = finalDataSet.get(0);
        for (int i = 0; i < firstTransaction.length - 1; i++) {

listOfAllDiscreteValuesBasedOnAttributeNumber.put(i, new
ArrayList<String>());
        }
        for (String[] transactionDataset : finalDataSet) {
            for (int i = 0; i < transactionDataset.length - 1;
i++) {

                String value = transactionDataset[i];
                if (value.equals("?"))
                    continue;
                if (!differentAttributesList.contains(value)) {

listOfAllDiscreteValuesBasedOnAttributeNumber.get(i).add(value);

                    differentAttributesList.add(value);
                }
            }
        }
    }

```

```
    }  
}
```

class DecisionTreeComparator implements

```
Comparator<String[]> {
```

```
    int i;
```

```
    public DecisionTreeComparator(int i) {
```

```
        this.i = i;
```

```
    }
```

```
    public int compare(String[] string1, String[] string2) {
```

```
        return string1[i].compareTo(string2[i]);
```

```
    }
```

```
}
```

```
package com.es589;
```

```
import java.util.ArrayList;
```

```
import java.util.HashMap;
```

```
import java.util.List;
```

```
public class DecisionTreeServiceImpl {
```

```
    HashMap<Integer, ArrayList<String>>
```

```
listOfAllDiscreteValuesBasedOnAttributeNumber;
```

```
    List<String> outputList = new ArrayList<>();
```

```
    String output1, output2;
```

```
    List<String[]> trainingDataSets;
```

```

List<String[]> testingDataSet;
DecisionTreeNode decisionTreeNode;
double trainingTimeForExcecution = 0;
double precisionValue;
double recallValue;
double fscoreValue;
double accuracyValue;
int numberOfNodes;

    public DecisionTreeServiceImpl(ArrayList<String[]>
trainingDataSetAfterPreProcessing, ArrayList<String[]>
testData,
        List<Integer> numberOfTotalOutputs,
List<String> outputList,
        HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber,
        ArrayList<Integer>
remainingAttributesToBeAddedInTree) {
        trainingTimeForExcecution =
System.currentTimeMillis();

        this.trainingDataSets =
trainingDataSetAfterPreProcessing;
        this.output1 = outputList.get(0);
        this.output2 = outputList.get(1);
        this.testingDataSet = testData;

```



```

        this.listOfAllDiscreteValuesBasedOnAttributeNumber
= listOfAllDiscreteValuesBasedOnAttributeNumber;
        int totalNumberOfOutput1, totalNumberOfOutput2;
        totalNumberOfOutput1 =
numberOfTotalOutputs.get(0);
        totalNumberOfOutput2 =
numberOfTotalOutputs.get(1);
        decisionTreeNode = new DecisionTreeNode();

        double prob1 = totalNumberOfOutput1 /
(totalNumberOfOutput1 + totalNumberOfOutput2 + 0.0);
        double prob2 = totalNumberOfOutput2 /
(totalNumberOfOutput1 + totalNumberOfOutput2 + 0.0);
        decisionTreeNode.entropyValue = -1 *
calculate_PLogP(prob1) - 1 * calculate_PLogP(prob2);
        decisionTreeNode.dataSet =
trainingDataSetAfterPreProcessing;
        decisionTreeNode.numberofInstancesofOutput1 =
totalNumberOfOutput1;
        decisionTreeNode.numberofInstancesofOutput2 =
totalNumberOfOutput2;

decisionTreeNode.remainingAttributesToBeAddedinDecisionTr
ee = remainingAttributesToBeAddedInTree;
        buildingDecisionTree(decisionTreeNode);

```

```
        trainingTimeForExcecution =  
(System.currentTimeMillis() - trainingTimeForExcecution) /  
1000.0;
```

```
        decisionTreeAnalysis();  
    }
```

```
    public void decisionTreeAnalysis() {  
        int correctClassificationOutput = 0,  
incorrectClassificationOutput = 0;  
        int tf_TruePositive = 0, fp_falsePositive = 0,  
fn_falseNegative = 0;  
        for (String[] transactionDataSet : testingDataSet) {  
            int predictedValue =  
DecisionTreeNode.predictionOfOutput(decisionTreeNode,  
transactionDataSet,  
  
listOfAllDiscreteValuesBasedOnAttributeNumber),  
                actualValue =  
transactionDataSet[transactionDataSet.length -  
1].equals(output1) ? 1 : 2;  
            if (predictedValue == actualValue) {  
                correctClassificationOutput++;  
            } else {  
                incorrectClassificationOutput++;  
            }  
        }  
    }
```

```

        if (predictedValue == 1 && actualValue == 1)
            tf_TruePositive++;
        else if (predictedValue == 1 && actualValue ==
2)
            fn_falseNegative++;
        else if (predictedValue == 2 && actualValue ==
1)
            fp_falsePositive++;
    }
    precisionValue = tf_TruePositive / (tf_TruePositive +
fp_falsePositive + 0.0);
    recallValue = tf_TruePositive / (tf_TruePositive +
fn_falseNegative + 0.0);
    fscoreValue = 2 * precisionValue * recallValue /
(precisionValue + recallValue);
    accuracyValue = (correctClassificationOutput)
        / (correctClassificationOutput +
incorrectClassificationOutput + 0.0);

    numberOfNodes = 0;
    numberOfNodesOfDecsionTree(decisionTreeNode);
}

public void printAnalysisofdecisionTree() {
    System.out.println("No of nodes in tree = " +
numberOfNodes);

```

```
        System.out.println("Accuracy=" + accuracyValue +  
"\nPrecision=" + precisionValue + "\nRecall=" + recallValue  
        + "\nF-Score=" + fscoreValue);
```

```
    }
```

```
    public void printExecutionTimeOfDecisionTree() {  
        System.out.println("TrainingTime in secs:" +  
trainingTimeForExcecution);  
    }
```

```
    public void  
numberOfNodesOfDecsionTree(DecisionTreeNode root) {  
        if (root == null)  
            return;  
        numberOfNodes++;  
        if (root.leafNode)  
            return;  
        for (DecisionTreeNode n : root.childNodes)  
            numberOfNodesOfDecsionTree(n);  
    }
```

```
    private static double calculate_PLogP(double x) {  
        return x == 0 ? 0 : x * Math.log(x);  
    }
```

```

private void buildingDecisionTree(DecisionTreeNode root)
{
    if (root == null)
        return;
    if
(root.remainingAttributesToBeAddedinDecisionTree.size() ==
1) {
        root.leafNode = true;
        root.numberOfClassificationNodes =
root.numberOfInstancesofOutput1 >=
root.numberOfInstancesofOutput2 ? 1
            : 2;
    } else if (root.numberOfInstancesofOutput1 == 0 ||
root.numberOfInstancesofOutput2 == 0
        || root.dataSet.size() == 0) {
        root.leafNode = true;
        root.numberOfClassificationNodes =
root.numberOfInstancesofOutput1 == 0 ? 2 : 1;
    } else {
        // find split attribute which gives max gain
        root.childNodes =
splittingAttributeUsingTheSelectionMetrics(root);
        ArrayList<String> discreteValuesOfThisAttribute
= listOfAllDiscreteValuesBasedOnAttributeName
.get(root.currentAttributeName_ColumnNumber);

```

```

        for (int j = 0; j <
discreteValuesOfThisAttribute.size(); j++) {
            root.childNodes[j].dataSet = new
ArrayList<>();

root.childNodes[j].remainingAttributesToBeAddedinDecisionTr
ee = new ArrayList<>();
            for (int rem :
root.remainingAttributesToBeAddedinDecisionTree) {
                if (rem !=
root.currentAttributeNumber_ColumnNumber)

root.childNodes[j].remainingAttributesToBeAddedinDecisionTr
ee.add(rem);
            }
            String curr =
discreteValuesOfThisAttribute.get(j);
            for (String[] s : root.dataSet) {
                if
(s[root.currentAttributeNumber_ColumnNumber].equals(curr))
{
                    root.childNodes[j].dataSet.add(s);
                }
            }
            buildingDecisionTree(root.childNodes[j]);
        }
    }

```

```
}
```

```
    public DecisionTreeNode[]  
    splittingAttributeUsingTheSelectionMetrics(DecisionTreeNode  
    decisionTreeNode) {  
        double maximumGain = -1.0;  
        double maximumGini = -1.0;  
        DecisionTreeNode[] nodeToBeSelected = null;  
        double gini_Index = 0 ;  
        for (int i :  
    decisionTreeNode.remainingAttributesToBeAddedinDecisionTr  
    ee) {  
            ArrayList<String>  
    listOfDiscreteValuesOfThisAttribute =  
    listOfAllDiscreteValuesBasedOnAttributeNumber.get(i);  
            DecisionTreeNode[] childNode = new  
    DecisionTreeNode[listOfDiscreteValuesOfThisAttribute.size()];  
            for (int j = 0; j <  
    listOfDiscreteValuesOfThisAttribute.size(); j++) {  
                String present =  
    listOfDiscreteValuesOfThisAttribute.get(j);  
                childNode[j] = new DecisionTreeNode();  
                for (String[] tem :  
    decisionTreeNode.dataSet) {  
                    if (tem[i].equals(present)) {  
                        if (tem[tem.length -  
    1].equals(output1))
```

```

childNode[j].numberOfInstancesofOutput1++;
                    else

childNode[j].numberOfInstancesofOutput2++;
                    }
                }
            }
            int totalValue = decisionTreeNode.dataSet.size();
            double gain = decisionTreeNode.entropyValue;
            for (int j = 0; j <
listOfDiscreteValuesOfThisAttribute.size(); j++) {
                int outputClass1 =
childNode[j].numberOfInstancesofOutput1, outputClass2 =
childNode[j].numberOfInstancesofOutput2;
                if (outputClass1 == 0 && outputClass2 ==
0)

                    continue;
                double prob1 = outputClass1 / (outputClass1
+ outputClass2 + 0.0), prob2 = outputClass2 / (outputClass1 +
outputClass2 + 0.0);
                childNode[j].entropyValue = -1 *
calculate_PLogP(prob1) + -1 * calculate_PLogP(prob2);
                gini_Index = 1 - Math.pow(prob1, 2) -
Math.pow(prob2, 2);
                // gain -= ((outputClass1 + outputClass2) /
(totalValue + 0.0)) * childNode[j].entropyValue;

```



```

        }
//        if (gain > maximumGain) {
//
decisionTreeNode.currentAttributeNumber_ColumnNumber = i;
//        maximumGain = gain;
//        nodeToBeSeleceted = childNode;
//    }
    if(gini_Index>maximumGini) {

decisionTreeNode.currentAttributeNumber_ColumnNumber = i;
        maximumGini = gini_Index;
        nodeToBeSeleceted = childNode;

    }

}
return nodeToBeSeleceted;
}
}

```

```
package com.es589;
```

```

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Random;

```

```

public class RandomForestAlgorithm {
    int numberOfTrees;

```

```

        List<String> outputList = new ArrayList<>();
        HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber;
        String output1, output2;
        ArrayList<String[]> trainingDataSetAfterPreProcessing;
        ArrayList<String[]> testingDataSetAfterPreProcessing;
        double trainingTimeForExcecution = 0;
        double precisionValue;
        double recallValue;
        double fscoreValue;
        double accuracyValue;
        DecisionTreeServiceImpl[] decisionTree;

        public RandomForestAlgorithm(int numberOfTrees, double
attributesFraction, double trainingFraction,
        ArrayList<String[]>
trainingDataSetAfterPreProcessing, ArrayList<String[]>
testingDataSetAfterPreProcessing,
        List<Integer> numberOfTotalOutputs,
List<String> outputList,
        HashMap<Integer, ArrayList<String>>
listOfAllDiscreteValuesBasedOnAttributeNumber) {

            this.numberOfTrees = numberOfTrees;
            this.trainingDataSetAfterPreProcessing =
trainingDataSetAfterPreProcessing;
            this.output1 = outputList.get(0);

```

```
        this.output2 = outputList.get(1);
        this.testingDataSetAfterPreProcessing =
testingDataSetAfterPreProcessing;
        this.listOfAllDiscreteValuesBasedOnAttributeNumber
= listOfAllDiscreteValuesBasedOnAttributeNumber;
```

```
        int numberOfAttributes =
trainingDataSetAfterPreProcessing.get(0).length - 1,
        numberOfTrainingInstances =
trainingDataSetAfterPreProcessing.size();
        int numberOfRandomInstances = (int)
(numberOfTrainingInstances * trainingFraction);
        int numberOfRandomAttributes = (int)
(numberOfAttributes * attributesFraction);
```

```
        String[][] transactionDataSetInArray = new
String[numberOfTrainingInstances][];
        int q = 0;
        for (String[] transaction :
trainingDataSetAfterPreProcessing) {
            transactionDataSetInArray[q++] = transaction;
        }
```

```
        trainingTimeForExcecution =
System.currentTimeMillis();
```

```
        Random random = new Random();
```

```

        decisionTree = new
DecisionTreeServiceImpl[numberOfTrees];
        for (int i = 0; i < numberOfTrees; i++) {
            ArrayList<Integer> remainingAttrributes = new
ArrayList<>();
            for (int j = 0; j < numberOfRandomAttributes;
j++) {
                int ran =
random.nextInt(numberOfAttributes);
                if (remainingAttrributes.contains(ran))
                    j--;
                else
                    remainingAttrributes.add(ran);
            }
            ArrayList<String[]> randomDataSet = new
ArrayList<>();
            for (int j = 0; j < numberOfRandomInstances;
j++) {

randomDataSet.add(transactionDataSetInArray[random.nextInt(
numberOfTrainingInstances)]);
            }
            decisionTree[i] = new
DecisionTreeServiceImpl(randomDataSet,
testingDataSetAfterPreProcessing,
                            numberOfTotalOutputs, outputList,
listOfAllDiscreteValuesBasedOnAttributeNumber,

```

```

        remainingAttributes);
    }

    trainingTimeForExcecution =
(System.currentTimeMillis() - trainingTimeForExcecution) /
1000.0;

    randomForestAnalysis();
}

public void randomForestAnalysis() {
    if (decisionTree == null)
        return;
    int correctClassificationOutput = 0,
incorrectClassificationOutput = 0;
    int tf_TruePositive = 0, fp_FalsePositive = 0,
fn_FalseNegative = 0;
    for (String[] s : testingDataSetAfterPreProcessing) {
        int temp1 = 0, temp2 = 0;
        for (int i = 0; i < numberOfTrees; i++) {
            if
(DecisionTreeNode.predictionOfOutput(decisionTree[i].decision
TreeNode, s,

listOfAllDiscreteValuesBasedOnAttributeNumber) == 1)
                temp1++;
            else

```

```

        temp2++;
    }
    int predictedValue = temp1 >= temp2 ? 1 : 2,
        actualValue = s[s.length -
1].equals(output1) ? 1 : 2;
    if (predictedValue == actualValue)
        correctClassificationOutput++;
    else
        incorrectClassificationOutput++;
    if (predictedValue == 1 && actualValue == 1)
        tf_TruePositive++;
    else if (predictedValue == 1 && actualValue ==
2)
        fn_FalseNegative++;
    else if (predictedValue == 2 && actualValue ==
1)
        fp_FalsePositive++;
    }
    precisionValue = tf_TruePositive / (tf_TruePositive +
fp_FalsePositive + 0.0);
    recallValue = tf_TruePositive / (tf_TruePositive +
fn_FalseNegative + 0.0);
    fscoreValue = 2 * precisionValue * recallValue /
(precisionValue + recallValue);
    accuracyValue = (correctClassificationOutput) /
(correctClassificationOutput + incorrectClassificationOutput +
0.0);

```

```

        System.out.println("Training Time in secs = " +
trainingTimeForExcecution);
        System.out.println("Accuracy=" + accuracyValue +
"\nPrecision=" + precisionValue + "\nRecall=" + recallValue
        + "\nF-Score=" + fscoreValue);
    }
}

```

```

package com.es589;

```

```

public class ReducedErrorPruningOnDecisionTree {
    DecisionTreeServiceImpl decisionTree;
    double maximumAccuracy;
    double accuracyOfDecsionTreeBeforeREP;

    public
ReducedErrorPruningOnDecisionTree(DecisionTreeServiceImpl
decisionTree) {
        this.decisionTree = decisionTree;
        accuracyOfDecsionTreeBeforeREP =
decisionTree.accuracyValue;
        maximumAccuracy =
accuracyOfDecsionTreeBeforeREP;
        double trainingTimeStart =
System.currentTimeMillis();

    pruneDecsionTreeNode(decisionTree.decisionTreeNode);
}
}

```

```

        double trainingTimeEnd =
System.currentTimeMillis();
        double totalTimeTaken = (trainingTimeEnd -
trainingTimeStart) / 1000.0;
        System.out.println("Training Time in seconds=" +
totalTimeTaken);
    }

    private void pruneDecsionTreeNode(DecisionTreeNode
decisionTreeNode) {
        if (decisionTreeNode == null)
            return;
        decisionTreeNode.leafNode = true;
        decisionTreeNode.numberOfClassificationNodes =
decisionTreeNode.numberOfInstancesofOutput1 >=
decisionTreeNode.numberOfInstancesofOutput2 ? 1 : 2;
        decisionTree.decisionTreeAnalysis();
        if (decisionTree.accuracyValue > maximumAccuracy)
        {
            maximumAccuracy =
decisionTree.accuracyValue;
            return;
        }
        decisionTreeNode.leafNode = false;
        for (DecisionTreeNode treeNode :
decisionTreeNode.childNodes) {
            if (treeNode.leafNode)

```



```
        continue;
    pruneDecsionTreeNode(treeNode);
}

}

}
```