DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

Title: Introduction to WEKA using Java Based Approach

ARTIFICIAL INTELLIGENCE LAB
CSE 404

GREEN UNIVERSITY OF BANGLADESH

1 Objective(s)

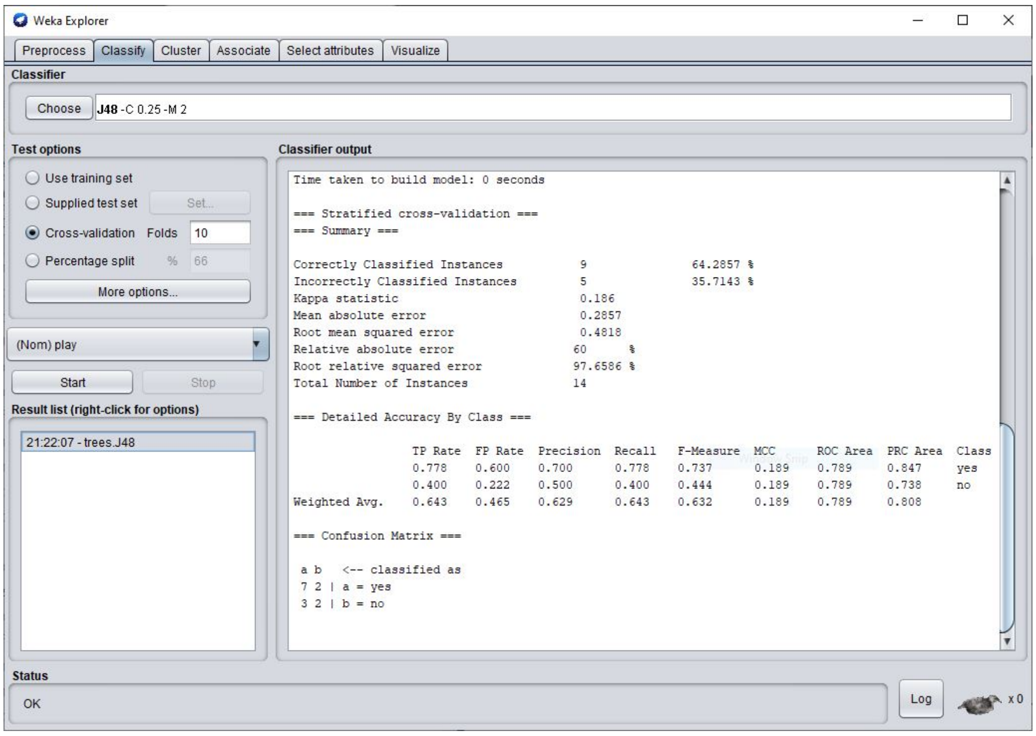
- To understand WEKA tool usage to analysis training data.
- To understand how to use Java programming to perform data analysis in WEKA.

2 Problem analysis

Here in WEKA we can perform analysis on training data using Java code directly. To do so we will first observe the classification algorithm- "Decision Tree" usage using GUI and then the Java implementation of the same process.

2.1 Classification Using Decision Tree Algorithm on WEKA GUI

First we will open the "*weather.numeric.arff*" file using WEKA explorer. After that we will select "Decision Tree" algorithm by following options - $Choose -> tree -> J48$. Now we will run this algorithm on the opened file by Clicking "Start" button. We can observe the result of running $J48$ as given in the figure 1 We can



Classifier: J48 -C 0.25 -M 2

Cross-validation Folds 10

% 66

(Nom) play

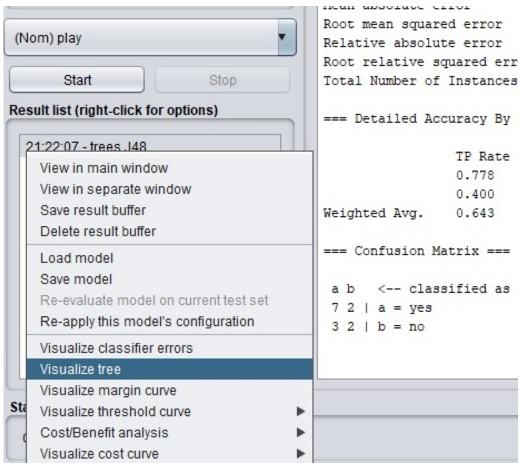
21:22:07 - trees.J48

OK

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0.186
Mean absolute error                      0.2857
Root mean squared error                  0.4818
Relative absolute error                 60      %
Root relative squared error             97.6586 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.778    0.600    0.700      0.778    0.737      0.189    0.789     0.847     yes
                 0.400    0.222    0.500      0.400    0.444      0.189    0.789     0.738     no
Weighted Avg.    0.643    0.465    0.629      0.643    0.632      0.189    0.789     0.808

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```

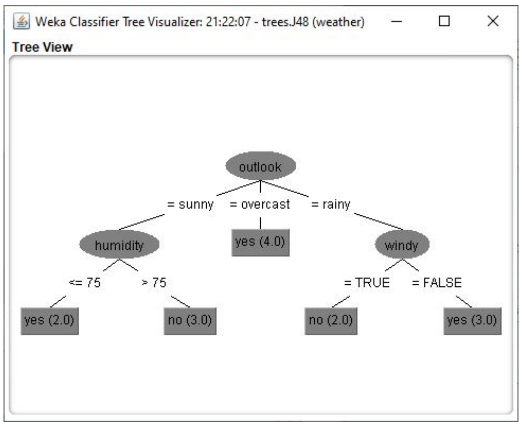
Figure 1: Output of J48 algorithm for classification

visualize the underlying decision tree generated by the $J48$ algorithm. In this case we first need to right click on $ResultList$ and select $Visualizetree$ option given in figure 3a. Also the figure 3b represents the tree generated by $J48$ algorithm.



```
Root mean squared error
Relative absolute error
Root relative squared err
Total Number of Instances

=== Detailed Accuracy By

               TP Rate
               0.778
               0.400
Weighted Avg.  0.643

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```

(a) Option for Decision Tree visualization



(b) Tree generated by J48 algorithm

Figure 2: Visualization of Decision Tree

2.2 Classification Using Decision Tree Algorithm - Java Implementation

We can implement *J*48 using java coding. For this purpose we open new project and use the java code given in the subsection 2.3. We must have to include the *weka.jar* file in *Libraries* of our project and resolve errors. When we run the java code for *J*48 Algorithm we will get the output given in the subsection 2.4.

2.3 Implementation in Java

```java
package testweka;

/**
 *
 * @author Jargis
 */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.functions.*;
import weka.classifiers.evaluation.NominalPrediction;
import weka.classifiers.rules.DecisionTable;
import weka.classifiers.rules.PART;
import weka.classifiers.trees.DecisionStump;
import weka.classifiers.trees.J48;
import weka.core.Debug.Random;
import weka.core.FastVector;
import weka.core.Instances;

public class TestWeka {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws Exception {
        simpleWekaTrain("C:\\Program Files\\Weka-3-8\\data\\weather.numeric.arff
            ");
        // TODO code application logic here
    }

    public static void simpleWekaTrain(String filepath) throws FileNotFoundException, Exception {
        try {
            //Reading training arff or csv file
            FileReader trainreader = new FileReader(filepath);
            Instances train = new Instances(trainreader);
            train.setClassIndex(train.numAttributes() - 1);
            //Instance of NN

            //Setting Parameters
            //MultilayerPerceptron mlp = new MultilayerPerceptron();
            //mlp.setLearningRate(0.3);
            //mlp.setMomentum(0.2);
            //mlp.setTrainingTime(500);
            //mlp.setHiddenLayers("4");
            //mlp.buildClassifier(train);
            J48 j48 = new J48();
            j48.setConfidenceFactor(0.25f);

            j48.buildClassifier(train);

            Evaluation eval = new Evaluation(train);
            eval.crossValidateModel(j48, train, 10, new Random(1));
            //eval.crossValidateModel(mlp, train, 10, new Random(1));
            //System.out.println(eval.errorRate()); //Printing Training Mean root squared Error
            System.out.println(eval.toSummaryString()); //Summary of Training

            /// Print Confusion Matrix
            System.out.println("=== Confusion Matrix ==="); //Summary of Training
            double conMat[][] = eval.confusionMatrix();
            int i, j;
            for (i = 0; i < conMat.length; ++i) {
                for (j = 0; j < conMat[i].length; ++j) {
                    System.out.print(conMat[i][j] + " ");
                }
                System.out.println("");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        // this lines of code can be used for saving training model and use the saved model for testing when required
        //weka.core.SerializationHelper.write("trained.model", mlp); // save tained model

        // use saved trained model for testing on data "Instance"
        //J48 classifier = (J48) weka.core.SerializationHelper.read(new FileInputStream(new File("ModelPath")));
        //eval.evaluateModel(classifier, data,plainText);
    }

}
```

2.4 Sample Input/Output (Compilation, Debugging & Testing)

```
Correctly Classified Instances          9               64.2857 %
Incorrectly Classified Instances        5               35.7143 %
Kappa statistic                         0.186
Mean absolute error                     0.2857
Root mean squared error                 0.4818
Relative absolute error                60      %
Root relative squared error            97.6586 %
Total Number of Instances              14

=== Confusion Matrix ===
7.0 2.0
3.0 2.0
```

3 Listing Accuracy Values

In the *J*47 classification algorithm we can see for fold number 10 we gain 64.2857% accuracy. We can change the fold number and observe the change the we get in accuracy values. A sample table containing the different fold number and corresponding accuracy value is given below-

Fold Number	Accuracy (%)
2	42.85
3	64.28
4	64.28
5	64.28

4 Discussion & Conclusion

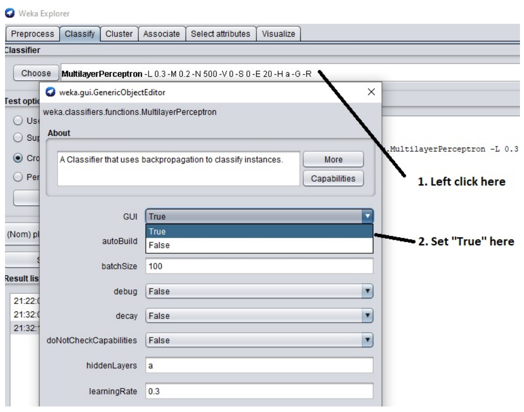
Based on the focused objective(s) to understand the use of WEKA tool and practice different classification algorithms. The java based implementation of WEKA can improve coding skills to manipulate and analysis data. The additional lab exercise will increase confidence towards the fulfilment of the objectives(s).

5 Lab Exercise (Submit as a report)

- Run Neural Network classification algorithm and evaluate the results of the method on a data file. Also make training model and use the training model to evaluate a testing file and compare accuracy values for different parameters.
  - Hint: For Neural Network you have to use *MultilayerPerceptron* algorithm which is reside in functions folder. to visualize the Neural Network follow the procedure in figure 3

6 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected for lab exercise.

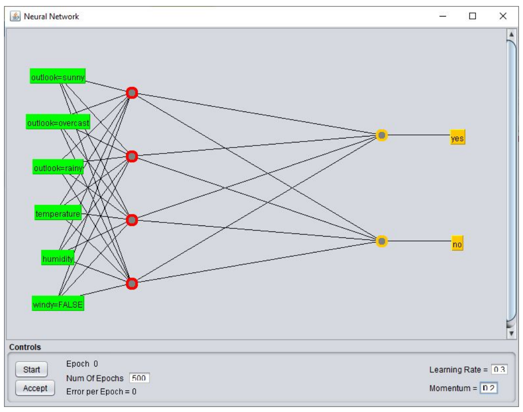


MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -G -R

weka.classifiers.functions.MultilayerPerceptron

A Classifier that uses backpropagation to classify instances.

GUI	True
autoBuild	
batchSize	100
debug	False
decay	False
doNotCheckCapabilities	False
hiddenLayers	a
learningRate	0.3



Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2

(a) Setting the options for Option for *MultilayerPerceptron* Algorithm

(b) Underlying Neural Network after using *MultilayerPerceptron* Algorithm

Figure 3: Visualization of Neural Network