Name: Kunal Sinha Roll no: 2K17/CO/164 Batch: CO-A3

# Experiment 4

## Aim:

A. Identify two data sets in which baseline classification is better then j48. Show the results using WEKA.
B. Calculate the sample mean and SD for any data set(at least 1500 instant and 6 classes) by setting the random number seed from 1 to 10 using WEKA.
C. Show the results for a handout(10%) over 10-cross validation using WEKA.

## Part A

Identify two data sets in which baseline classification is better then j48. Show the results using WEKA.

## Theory:

**Baseline Classification:** A baseline prediction algorithm provides a set of predictions that you can evaluate as you would any predictions for your problems, such as classification accuracy or RMSE. The scores from these algorithms provide the required point of comparison when evaluating all other machine learning algorithms on your problem. Once established, you can comment on how much better a given algorithm is as compared to the naive baseline algorithm, providing context on just how good a given method actually is.
The two most commonly used baseline algorithms are:

- Random Prediction Algorithm.
- Zero Rule Algorithm.

**J48 Tree Algorithm:** C4.5 (J48) is an algorithm used to generate a decision tree developed by Ross Quinlan mentioned earlier. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. It became quite popular after ranking #1 in the Top 10 Algorithms in Data Mining pre-eminent paper published by Springer LNCS in 2008.

## Output:

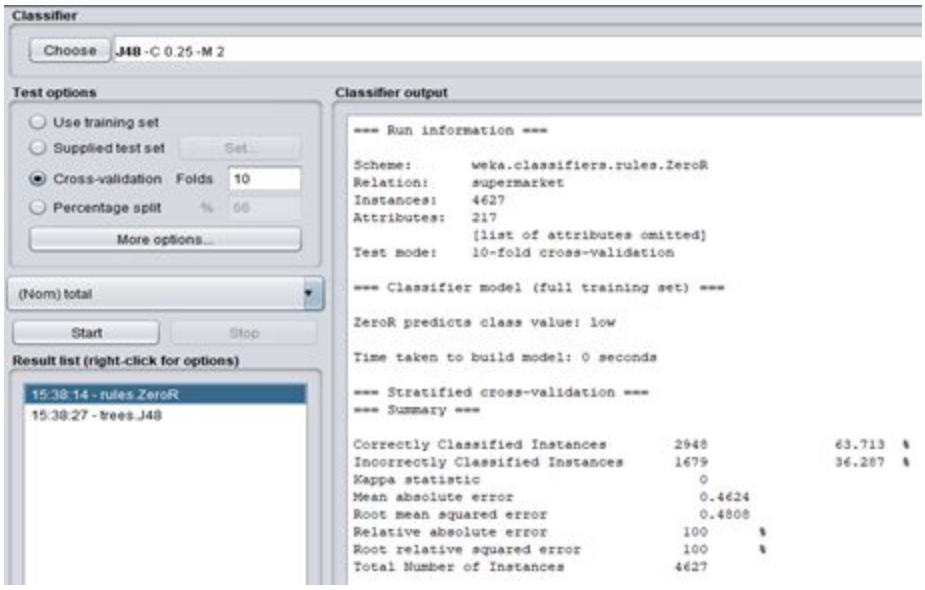
**Dataset 1:**

**Supermarket. arff**

1. **Zero R :**

   Accuracy 63.713

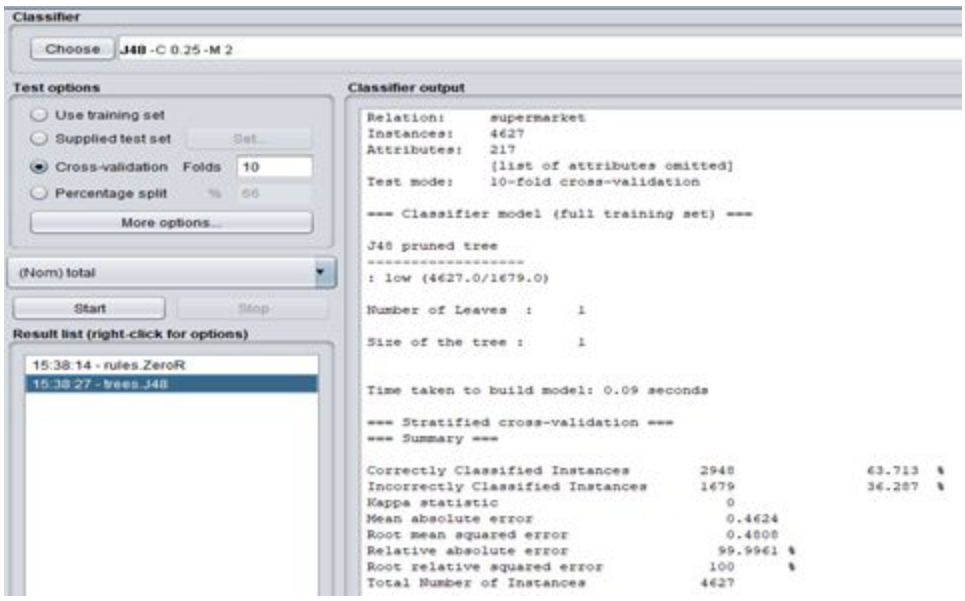
   Time is taken to build 0.01sec




2. **J48 :**

   Accuracy 63.713

   Time is taken to build 0.09sec

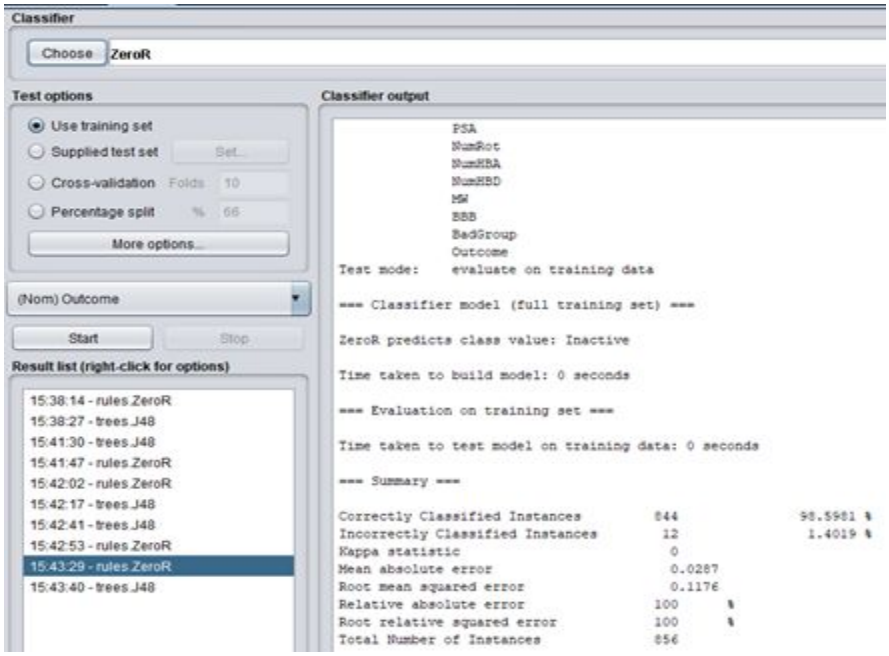


**Unbalanced. Arff**

1. **Zero R :**

   Accuracy 98.5981

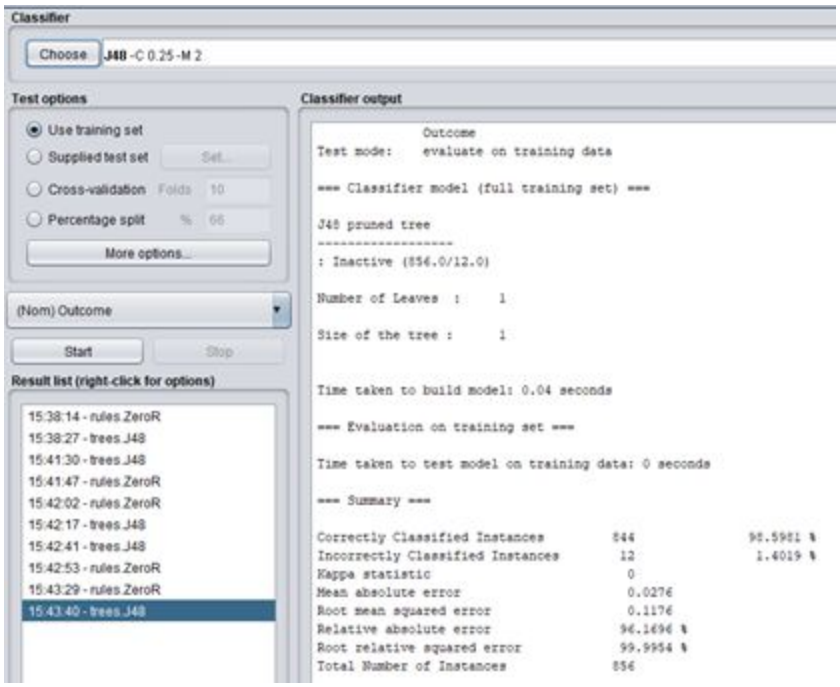
   Time is taken to build 0.01sec




2. **J48 :**

   Accuracy 98.5981

   Time is taken to build 0.04sec



## Part B

Calculate the sample mean and SD for any data set(at least 1500 instant and 6 classes) by setting the random number seed from 1 to 10 using WEKA

## Theory:

**Why is randomness important?**

It may be clear that reproducibility in machine learning is important, but how do we balance this with the need for randomness? There are both practical benefits for randomness and constraints that force us to use randomness.

Practically speaking, memory and time constraints have also forced us to 'lean' on randomness. Gradient Descent is one of the most popular and widely used algorithms for training machine learning models, however, computing the gradient step based on the entire dataset isn't feasible for large datasets and models. Stochastic Gradient Descent (SGD) only uses one or a mini-batch of randomly picked training samples from the training set to do the update for a parameter in a particular iteration.

While SGD might lead to a noisier error in the gradient estimate, this noise can actually encourage exploration to escape shallow local minima more easily. You can take this one step farther with simulated annealing, an extension of SGD, where the model purposefully take random steps in order to seek a better state.

Here are some important parts of the machine learning workflow where randomness appears:

1. **Data preparation**: in the case of a neural network, the shuffled batches will lead to different loss values across runs. This means your gradient values will be different across runs, and you will probably converge to a different local minima For specific types of data like time-series, audio, or text data plus specific types of models like LSTMs and RNNs, your data's input order can dramatically impact model performance.
2. **Data preprocessing**: over or upsampling data to address class imbalance involves randomly selecting an observation from the minority class with replacement. Upsampling can lead to overfitting because you're showing the model the same example multiple times.
3. **Cross-validation:** Both K-fold and Leave One Out Cross Validation (LOOCV) involve randomly splitting your data in order to evaluate the generalization performance of the model
4. **Weight initialization:** The initial weight values for machine learning models are often set to small, random numbers (usually in the range [-1, 1] or [0, 1]). Deep learning frameworks offer a variety of initialization methods from initializing with zeros to initializing from a normal distribution (see the Keras Initializers documentation as an example plus this excellent resource).

5. **Hidden layers in the network**: Dropout layers will randomly ignore a subset of nodes (each with a probability of being dropped, 1-p) during a particular forward or backwards pass. This leads to differences in layer activations even when the same input is used.
6. **Algorithms themselves**: Some models, such as random forest, are naturally dependent on randomness and others use randomness as a way of exploring the space.

These factors all contribute to variations across runs — making reproducibility very difficult even if you're working with the same model code and training data. Getting control over non-determinism and visibility into your experimentation process is crucial.

## Source Code:

```
// Generated with Weka 3.8.3
// This code is public domain and comes with no warranty.
// Timestamp: Tue Mar 02 15:57:51 IST 2021

package weka.classifiers;
import weka.core.Attribute;
import weka.core.Capabilities;
import weka.core.Capabilities.Capability;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.RevisionUtils;
import weka.classifiers.Classifier;
import weka.classifiers.AbstractClassifier;

public class WekaWrapper
  extends AbstractClassifier {
 /**
  * Returns only the toString() method.
  *
  * @return a string describing the classifier
  */
 public String globalInfo() {
   return toString();
 }
 /**
  * Returns the capabilities of this classifier.
  *
  * @return the capabilities
  */
 public Capabilities getCapabilities() {
```

```
  weka.core.Capabilities result = new weka.core.Capabilities(this);
  result.enable(weka.core.Capabilities.Capability.NOMINAL_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.NUMERIC_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.DATE_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.MISSING_VALUES);
  result.enable(weka.core.Capabilities.Capability.NOMINAL_CLASS);
  result.enable(weka.core.Capabilities.Capability.MISSING_CLASS_VALUES);

  result.setMinimumNumberInstances(0);
  return result;
}
/**
 * only checks the data against its capabilities.
 *
 * @param i the training data
 */
public void buildClassifier(Instances i) throws Exception {
  // can classifier handle the data?
  getCapabilities().testWithFail(i);
}
/**
 * Classifies the given instance.
 *
 * @param i the instance to classify
 * @return the classification result
 */
public double classifyInstance(Instance i) throws Exception {
  Object[] s = new Object[i.numAttributes()];
  for (int j = 0; j < s.length; j++) {
        if (!i.isMissing(j)) {
        if (i.attribute(j).isNominal())
        s[j] = new String(i.stringValue(j));
        else if (i.attribute(j).isNumeric())
        s[j] = new Double(i.value(j));
        }
  }
  // set class value to missing
  s[i.classIndex()] = null;
  return WekaClassifier.classify(s);
}
```

```
  /**
   * Returns the revision string.
   *
   * @return     the revision
   */
  public String getRevision() {
    return RevisionUtils.extract("1.0");
  }
  /**
   * Returns only the classnames and what classifier it is based on.
   *
   * @return a short description
   */
  public String toString() {
     return "Auto-generated classifier wrapper, based on weka.classifiers.trees.J48 (generated with
Weka 3.8.3).\n" + this.getClass().getName() + "/WekaClassifier";
  }
  /**
   * Runs the classfier from commandline.
   *
   * @param args the commandline arguments
   */
  public static void main(String args[]) {
    runClassifier(new WekaWrapper(), args);
  }
}
class WekaClassifier {
   public static double classify(Object[] i)
    throws Exception {
     double p = Double.NaN;
    p = WekaClassifier.N5fba94830(i);
    return p;
  }
  static double N5fba94830(Object []i) {
    double p = Double.NaN;
    if (i[1] == null) {
          p = 1;
    } else if (((Double) i[1]).doubleValue() <= 155.0) {
    p = WekaClassifier.N2688c4c11(i);
    } else if (((Double) i[1]).doubleValue() > 155.0) {
    p = WekaClassifier.N7fa0ebc929(i);
```

```
  }
  return p;
}
static double N2688c4c11(Object []i) {
  double p = Double.NaN;
  if (i[16] == null) {
        p = 2;
  } else if (((Double) i[16]).doubleValue() <= 91.4444) {
  p = WekaClassifier.N10924fa22(i);
  } else if (((Double) i[16]).doubleValue() > 91.4444) {
        p = 1;
  }
  return p;
}
static double N10924fa22(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 0;
  } else if (((Double) i[10]).doubleValue() <= 24.6667) {
  p = WekaClassifier.N2ce13ee13(i);
  } else if (((Double) i[10]).doubleValue() > 24.6667) {
  p = WekaClassifier.N383e78e426(i);
  }
  return p;
}
static double N2ce13ee13(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -1.89048) {
  p = WekaClassifier.N2774b9724(i);
  } else if (((Double) i[18]).doubleValue() > -1.89048) {
  p = WekaClassifier.N168bc6ca21(i);
  }
  return p;
}
static double N2774b9724(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.22266) {
```

```
  p = WekaClassifier.N473adf915(i);
  } else if (((Double) i[18]).doubleValue() > -2.22266) {
  p = WekaClassifier.N4557778b6(i);
  }
  return p;
}
static double N473adf915(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() <= 146.0) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() > 146.0) {
        p = 3;
  }
  return p;
}
static double N4557778b6(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() <= 2.55556) {
  p = WekaClassifier.N60e45ff67(i);
  } else if (((Double) i[10]).doubleValue() > 2.55556) {
  p = WekaClassifier.N62543c2013(i);
  }
  return p;
}
static double N60e45ff67(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.09121) {
  p = WekaClassifier.N62d9d5b18(i);
  } else if (((Double) i[18]).doubleValue() > -2.09121) {
        p = 4;
  }
  return p;
}
static double N62d9d5b18(Object []i) {
  double p = Double.NaN;
```

```
  if (i[1] == null) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() <= 129.0) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() > 129.0) {
  p = WekaClassifier.N730e482c9(i);
  }
  return p;
}
static double N730e482c9(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 128.0) {
  p = WekaClassifier.Nd1b82e910(i);
  } else if (((Double) i[0]).doubleValue() > 128.0) {
  p = WekaClassifier.N4fc2675311(i);
  }
  return p;
}
static double Nd1b82e910(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() <= 0.666667) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() > 0.666667) {
        p = 4;
  }
  return p;
}
static double N4fc2675311(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 0.333333) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 0.333333) {
  p = WekaClassifier.N631a474712(i);
  }
  return p;
```

```
}
static double N631a474712(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 4;
  } else if (((Double) i[0]).doubleValue() <= 216.0) {
        p = 4;
  } else if (((Double) i[0]).doubleValue() > 216.0) {
        p = 2;
  }
  return p;
}
static double N62543c2013(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 4;
  } else if (((Double) i[1]).doubleValue() <= 121.0) {
  p = WekaClassifier.N3309e64914(i);
  } else if (((Double) i[1]).doubleValue() > 121.0) {
  p = WekaClassifier.N1d4a278217(i);
  }
  return p;
}
static double N3309e64914(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 0;
  } else if (((Double) i[15]).doubleValue() <= -15.4444) {
        p = 0;
  } else if (((Double) i[15]).doubleValue() > -15.4444) {
  p = WekaClassifier.N6fadf78415(i);
  }
  return p;
}
static double N6fadf78415(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 2.94444) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 2.94444) {
```

```
  p = WekaClassifier.N5ebcb54916(i);
  }
  return p;
}
static double N5ebcb54916(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() <= 134.0) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() > 134.0) {
        p = 4;
  }
  return p;
}
static double N1d4a278217(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 4;
  } else if (((Double) i[10]).doubleValue() <= 7.88889) {
  p = WekaClassifier.N618da1cc18(i);
  } else if (((Double) i[10]).doubleValue() > 7.88889) {
  p = WekaClassifier.N611c6bae19(i);
  }
  return p;
}
static double N618da1cc18(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 0;
  } else if (((Double) i[0]).doubleValue() <= 43.0) {
    p = 0;
  } else if (((Double) i[0]).doubleValue() > 43.0) {
        p = 4;
  }
  return p;
}
static double N611c6bae19(Object []i) {
  double p = Double.NaN;
  if (i[17] == null) {
        p = 3;
```

```
  } else if (((Double) i[17]).doubleValue() <= 0.492526) {
        p = 3;
  } else if (((Double) i[17]).doubleValue() > 0.492526) {
  p = WekaClassifier.N5a319c3920(i);
  }
  return p;
}
static double N5a319c3920(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 82.0) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() > 82.0) {
        p = 3;
  }
  return p;
}
static double N168bc6ca21(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 0;
  } else if (((Double) i[15]).doubleValue() <= -4.77778) {
  p = WekaClassifier.N1a04861622(i);
  } else if (((Double) i[15]).doubleValue() > -4.77778) {
  p = WekaClassifier.N4ec1588624(i);
  }
  return p;
}
static double N1a04861622(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 0;
  } else if (((Double) i[5]).doubleValue() <= 2.77778) {
        p = 0;
  } else if (((Double) i[5]).doubleValue() > 2.77778) {
  p = WekaClassifier.N1847dfe323(i);
  }
  return p;
}
static double N1847dfe323(Object []i) {
```

```
  double p = Double.NaN;
  if (i[1] == null) {
        p = 0;
  } else if (((Double) i[1]).doubleValue() <= 115.0) {
        p = 0;
  } else if (((Double) i[1]).doubleValue() > 115.0) {
        p = 2;
  }
  return p;
}
static double N4ec1588624(Object []i) {
  double p = Double.NaN;
  if (i[7] == null) {
        p = 4;
  } else if (((Double) i[7]).doubleValue() <= 0.833336) {
  p = WekaClassifier.N6347a0225(i);
  } else if (((Double) i[7]).doubleValue() > 0.833336) {
        p = 6;
  }
  return p;
}
static double N6347a0225(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 115.0) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() > 115.0) {
        p = 4;
  }
  return p;
}
static double N383e78e426(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.17742) {
  p = WekaClassifier.N1ea000c727(i);
  } else if (((Double) i[18]).doubleValue() > -2.17742) {
  p = WekaClassifier.N5ae16efd28(i);
  }
```

```
  return p;
}
static double N1ea000c727(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 5.0) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 5.0) {
        p = 2;
  }
  return p;
}
static double N5ae16efd28(Object []i) {
  double p = Double.NaN;
  if (i[12] == null) {
        p = 0;
  } else if (((Double) i[12]).doubleValue() <= 24.4444) {
        p = 0;
  } else if (((Double) i[12]).doubleValue() > 24.4444) {
        p = 3;
  }
  return p;
}
static double N7fa0ebc929(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 5;
  } else if (((Double) i[15]).doubleValue() <= -2.0) {
  p = WekaClassifier.N3492558130(i);
  } else if (((Double) i[15]).doubleValue() > -2.0) {
        p = 6;
  }
  return p;
}
static double N3492558130(Object []i) {
  double p = Double.NaN;
  if (i[17] == null) {
        p = 5;
  } else if (((Double) i[17]).doubleValue() <= 0.385555) {
  p = WekaClassifier.N2a2acdec31(i);
```

```
  } else if (((Double) i[17]).doubleValue() > 0.385555) {
        p = 3;
  }
  return p;
 }
 static double N2a2acdec31(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 3;
  } else if (((Double) i[1]).doubleValue() <= 159.0) {
  p = WekaClassifier.N7bedd8d232(i);
  } else if (((Double) i[1]).doubleValue() > 159.0) {
        p = 5;
  }
  return p;
 }
 static double N7bedd8d232(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() <= 208.0) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() > 208.0) {
        p = 5;
  }
  return p;
 }
}
```

## Output:

| Seeds | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| accuracy | 95.098 | 93.725 | 96.078 | 94.902 | 94.705 | 92.549 | 93.137 | 94.117 | 93.137 | 95.490 |

**Mean: 94.29412**
**Standard Deviation: 1.148717**

## Part C

Show the results for a handout(10%) over 10-cross validation using WEKA

## Theory:

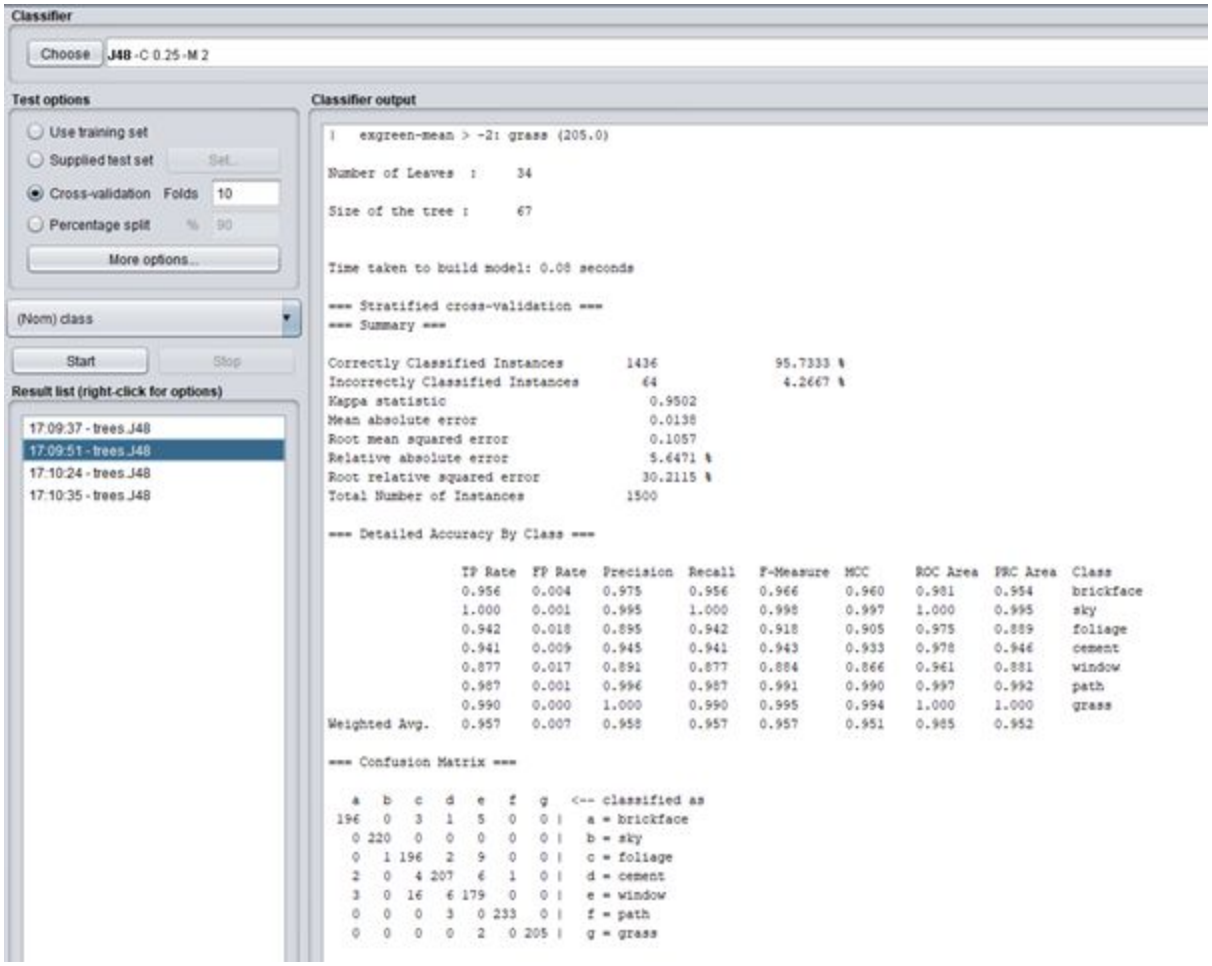
### Handout

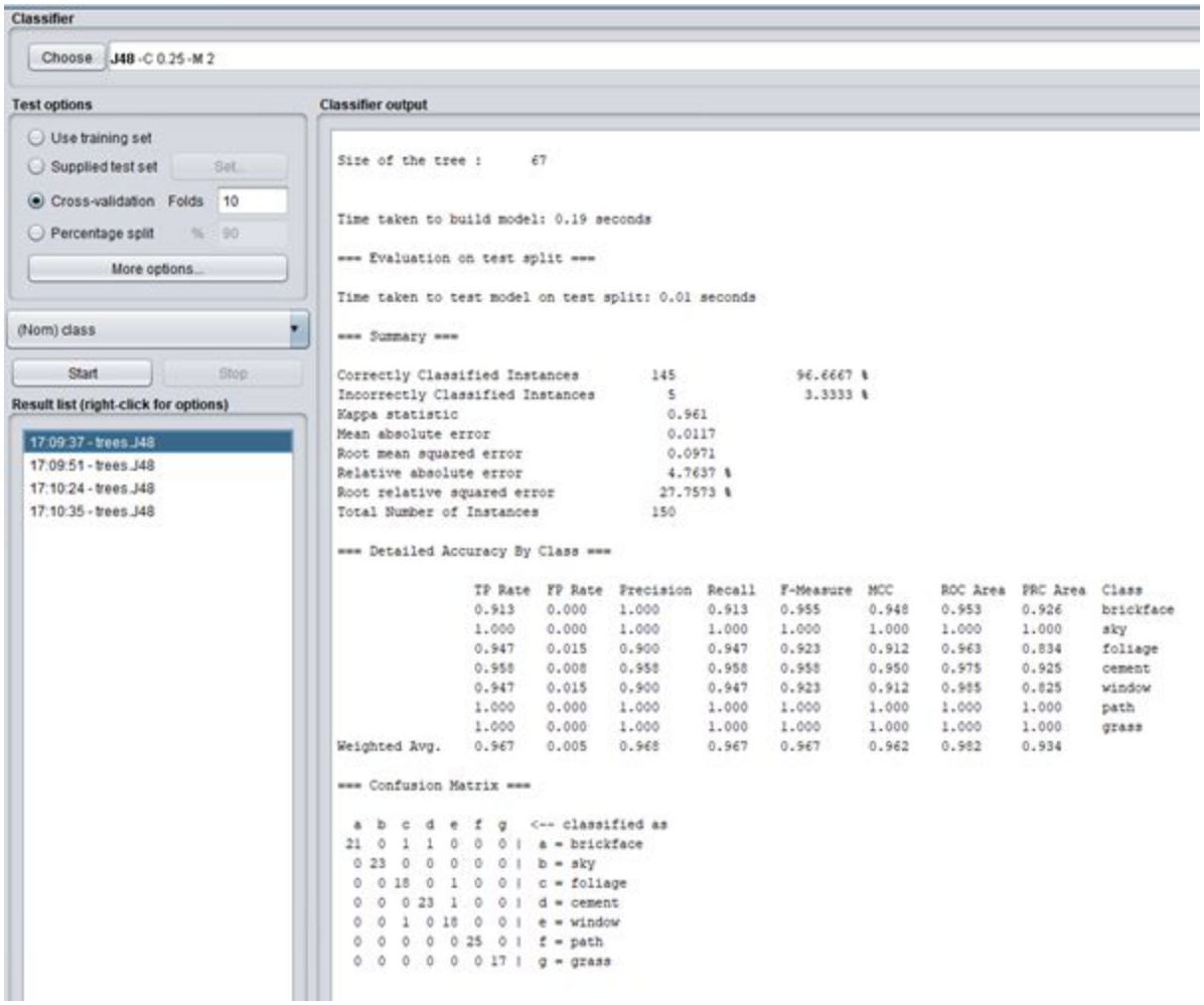
The handout is when you split up your dataset into a 'train' and 'test' set. The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data. A common split when using the handout method is using 80% of data for training and the remaining 20% of the data for testing.

### Cross-validation

Cross-validation or 'k-fold cross-validation' is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

## Output:






## Findings and Learnings :

- Many times we find that baselines match or outperform complex models, especially when the complex model has been chosen without looking at where the baseline fails. In addition, complex models are usually harder to deploy, which means measuring their lift over a simple baseline is a necessary precursor to the engineering efforts needed to deploy them.
- By carefully setting the random seed across your pipeline you can achieve reproducibility. The “seed” is a starting point for the sequence and the guarantee is that if you start from the same seed you will get the same sequence of numbers. Random seed helps in choosing different part of the dataset each time the algorithm works.
- Cross-validation is usually the preferred method because it gives your model the opportunity to train on multiple train-test splits. This gives you a better indication of how well your model will perform on unseen data. Handout, on the other hand, is dependent on just one train-test split. That makes the handout method score dependent on how the data is split into train and test sets. The handout method is good to use when you have a very large dataset, you’re on a time crunch, or you are starting to build an initial model in your data science project.