CP1407 ASSIGNMENT

Assignment 1 – Building and Testing Classifiers in WEKA

Student Name: Udaya Bhaskar Reddy Malkannagari

Student ID: 13368171

Task 1 – Classification for Wine

The dataset files include:
• train.arff (labeled training set, 1890 instances)
• test.arff (unlabeled test set, 810 instances)

This dataset is adapted from:
P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. *Modeling wine preferences by data mining from physicochemical properties.* In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.

This dataset contains data for 2700 white variants of the Portuguese "Vinho Verde" wine. For each variant, 11 chemical features were measured. Each of these is a numeric attribute. They are:
• fixed acidity
• volatile acidity
• citric acid
• residual sugar
• chlorides numeric
• free sulfur dioxide
• total sulfur dioxide
• density
• pH
• sulphates
• alcohol

Question #1:

Which attributes in the training set do not appear to have a "hump" distribution? Which

attributes appear to have outliers? (Do not worry too much about being precise here. The point is for you to inspect the data and interpret what you see.)
Based on the histogram, which attribute appears to be the most useful for classifying wine, and why?

Answer:

Which attributes in the training set do not appear to have a "hump" distribution?

Residual sugar

Explanation:

We can look at the visual representation of the data in separate histograms for each attribute and identify the skewness and symmetry in their distribution.

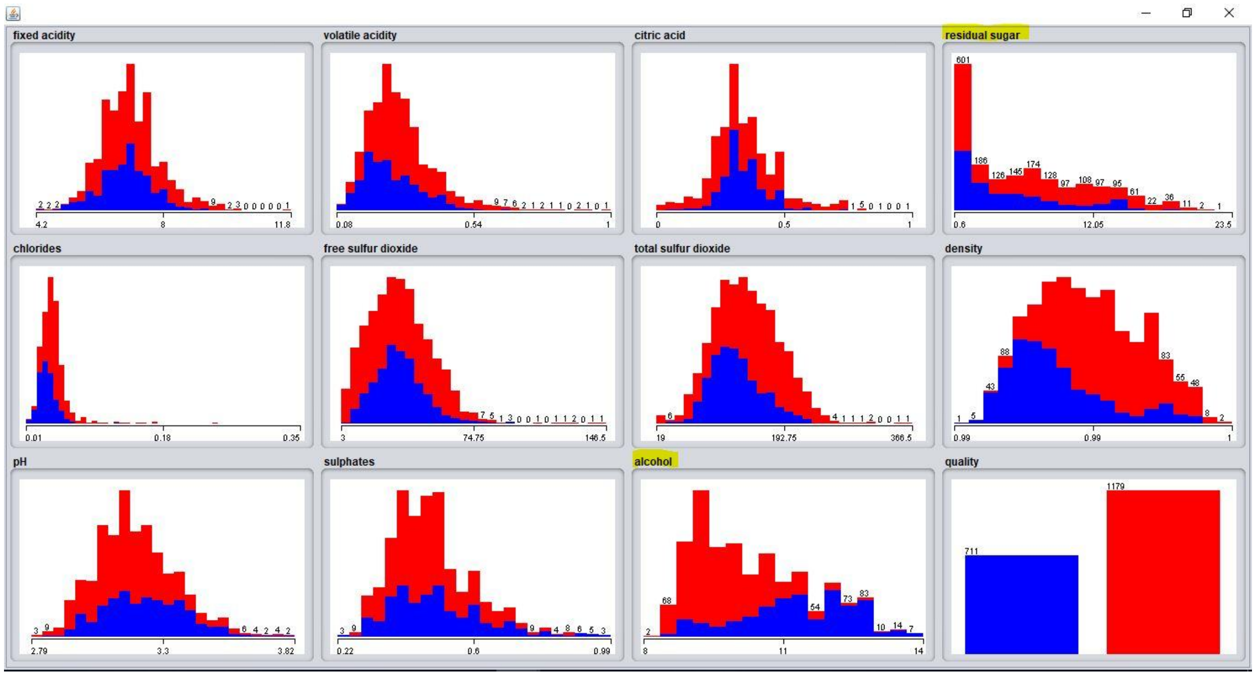


Figure 1: Visualizing the histograms of all the attributes in the training data

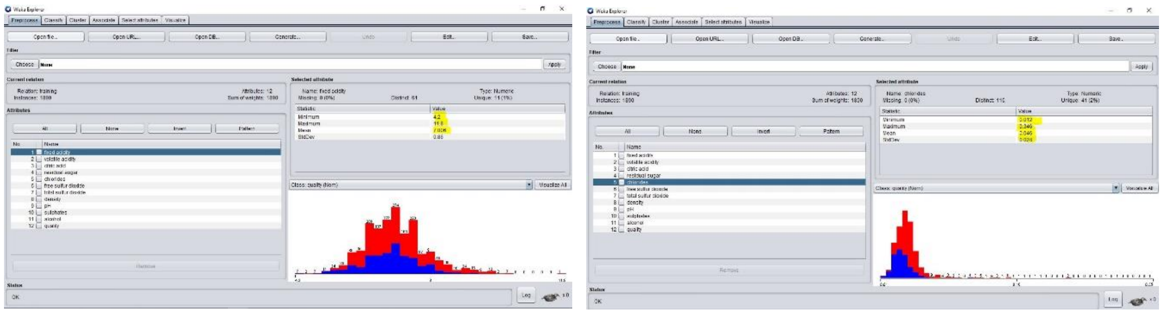
As we can observe, all the attribute histograms have a humped out graphic except for "Residual Sugar". All the attributes seem to have a unimodal distribution with possible outliers. At a first glance, most of the attributes appear slightly symmetrical in their alignment. However, the presence of outliers cannot be ignored.

The classifier attribute "Quality" displays two bars, one in Blue and the other in Red. Blue indicate the count of good quality wine and red indicates the count of bad quality wine.

We now take the example of the first attribute “fixed acidity” which has a mean of 7.006 with a minimum value of 4.2 and a maximum of 11.8. This makes it appear more symmetric, even though we notice an outlier to the right. (Figure-2)

We can also look at another attribute “Chlorides” which has a mean of 0.046 with a minimum value of 0.012 and a maximum of 0.346. The presence of outliers all along an extended right tail makes for an uneven distribution of data. (Figure -2)

However, we notice a humped distribution is all these attributes except ‘Residual Sugar’. ‘Alcohol’ also does not seem to have an elevated hump as we observe a higher peak at one of the nodes towards the left. Even the attribute ‘Chlorides’ has a high peak which cannot be equated to a hump (Figure 2), but these assumptions can only be subjective in nature.



Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

Name: fixed acidity
Missing: 0 (0%)
Distinct: 61
Type: Numeric
Unique: 11 (1%)

Statistic	Value
Minimum	4.2
Maximum	11.8
Mean	7.006
StdDev	0.85

Name: chlorides
Missing: 0 (0%)
Distinct: 116
Type: Numeric
Unique: 41 (2%)

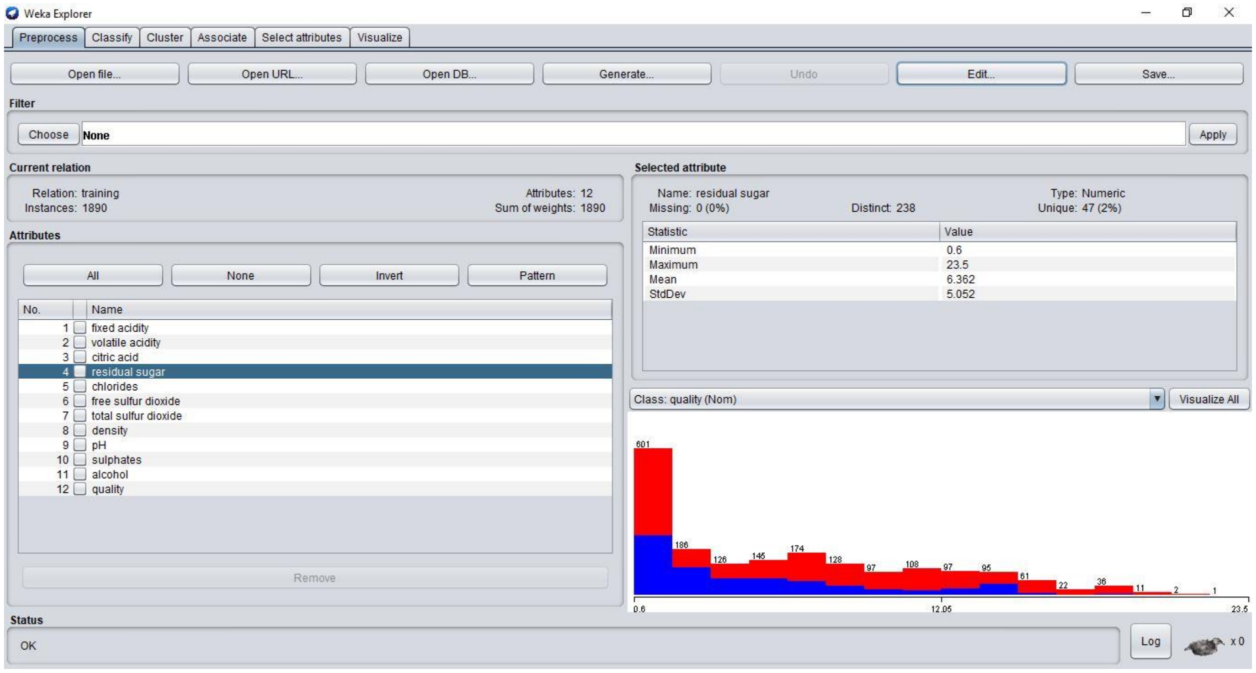
Statistic	Value
Minimum	0.012
Maximum	0.346
Mean	0.046
StdDev	0.024

Class: quality (Nom)

Figure 2: Showing the mean, min and max values of “Fixed Acidity” and “Chlorides”

Residual Sugar:

Now we look at the histogram for the attribute “Residual Sugar”



Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

No.	Name
1	fixed acidity
2	volatile acidity
3	citric acid
4	residual sugar
5	chlorides
6	free sulfur dioxide
7	total sulfur dioxide
8	density
9	pH
10	sulphates
11	alcohol
12	quality

Name: residual sugar
Missing: 0 (0%)
Distinct: 238
Type: Numeric
Unique: 47 (2%)

Statistic	Value
Minimum	0.6
Maximum	23.5
Mean	6.362
StdDev	5.052

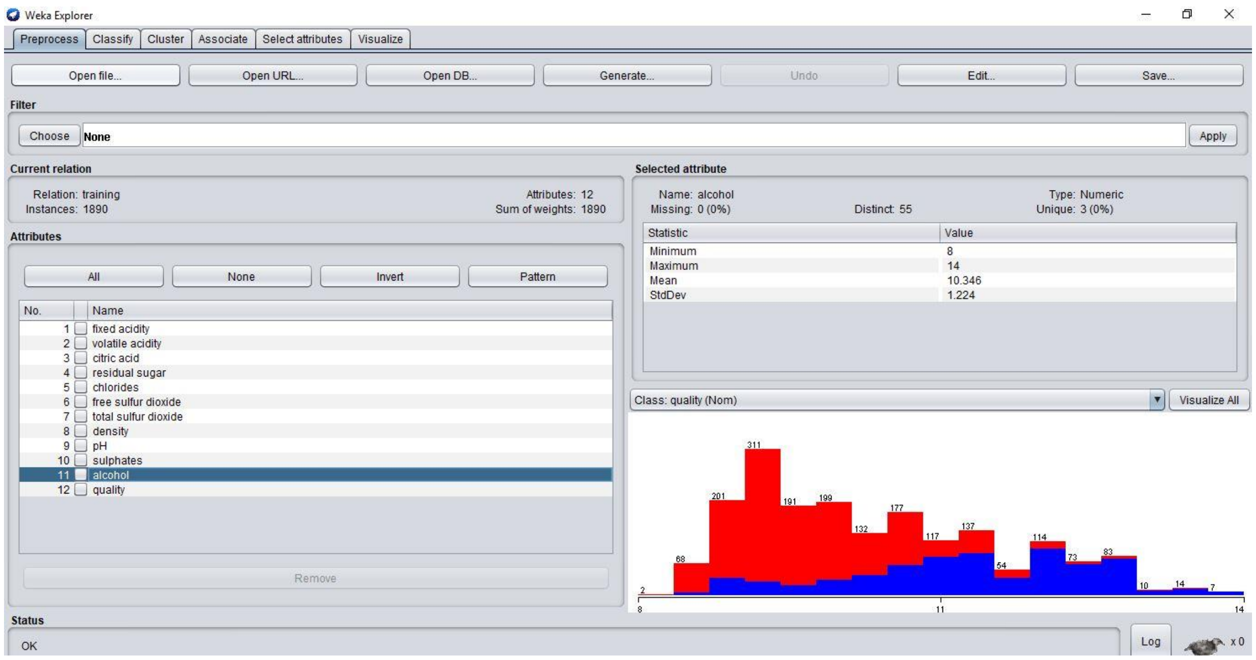
Class: quality (Nom)

Figure 3: Histogram for the attribute "Residual Sugar"

As we can observe, the histogram does not have a hump and goes from a high frequency (of quality) at the left to a very long tail on the right with a maximum value of 23.5. The high frequency bar at the left indicates more number of good and bad quality wines with a minimum value for residual sugar. Despite the long tail to the right, we can observe that there are no outliers under this attribute.

Alcohol:

Now we look at the histogram for the attribute "Alcohol"



Filter: None

Current relation
Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

Selected attribute
Name: alcohol
Missing: 0 (0%)
Distinct: 55
Type: Numeric
Unique: 3 (0%)

Statistic	Value
Minimum	8
Maximum	14
Mean	10.346
StdDev	1.224

Attributes

No.	Name
1	fixed acidity
2	volatile acidity
3	citric acid
4	residual sugar
5	chlorides
6	free sulfur dioxide
7	total sulfur dioxide
8	density
9	pH
10	sulphates
11	alcohol
12	quality

Class: quality (Nom)

Status
OK

Figure 4: Histogram for the attribute "Alcohol"

We can observe that there is a consistent distribution of data throughout the graphic. There is a peak around the 9 to 10 range of the alcohol percentage in the wine. From a subjective point of view, this can be treated as a hump, but the peak is a singular node and the data appears spread out around this high bar. We can also notice the lack of any outliers in the data distribution even though it has a slightly extended tail to the right.

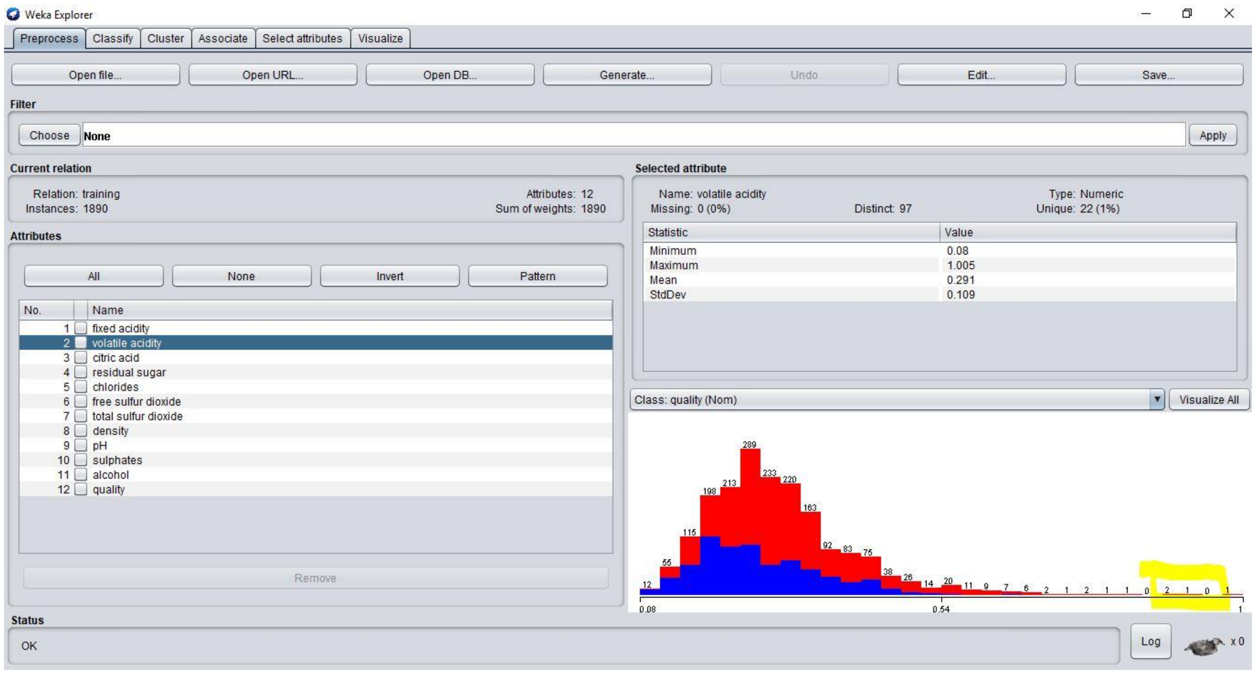
This attribute seems to be useful in classifying the wine as there is a clear visual pattern in its alignment.

Which attributes appear to have outliers?

Fixed Acidity, Volatile Acidity, Citric Acid, Chlorides, Free Sulfur Dioxide, Total Sulfur Dioxide

All these attributes seem to have outliers with a few points that are significantly removed from the rest of the data. We can also observe that some attributes seem to have an extended tail which signify the continuity of the data. This can be seen in the case of “pH” and “Sulphates” where there is a prolonged tail to the right (Figure 1).

We can take look at the histogram for “Volatile Acidity” to highlight the presence of outliers.



Filter: None

Current relation

Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

Selected attribute

Name: volatile acidity
Missing: 0 (0%)
Distinct: 97
Type: Numeric
Unique: 22 (1%)

Statistic	Value
Minimum	0.08
Maximum	1.005
Mean	0.291
StdDev	0.109

Attributes

No.	Name
1	fixed acidity
2	volatile acidity
3	citric acid
4	residual sugar
5	chlorides
6	free sulfur dioxide
7	total sulfur dioxide
8	density
9	pH
10	sulphates
11	alcohol
12	quality

Status

OK

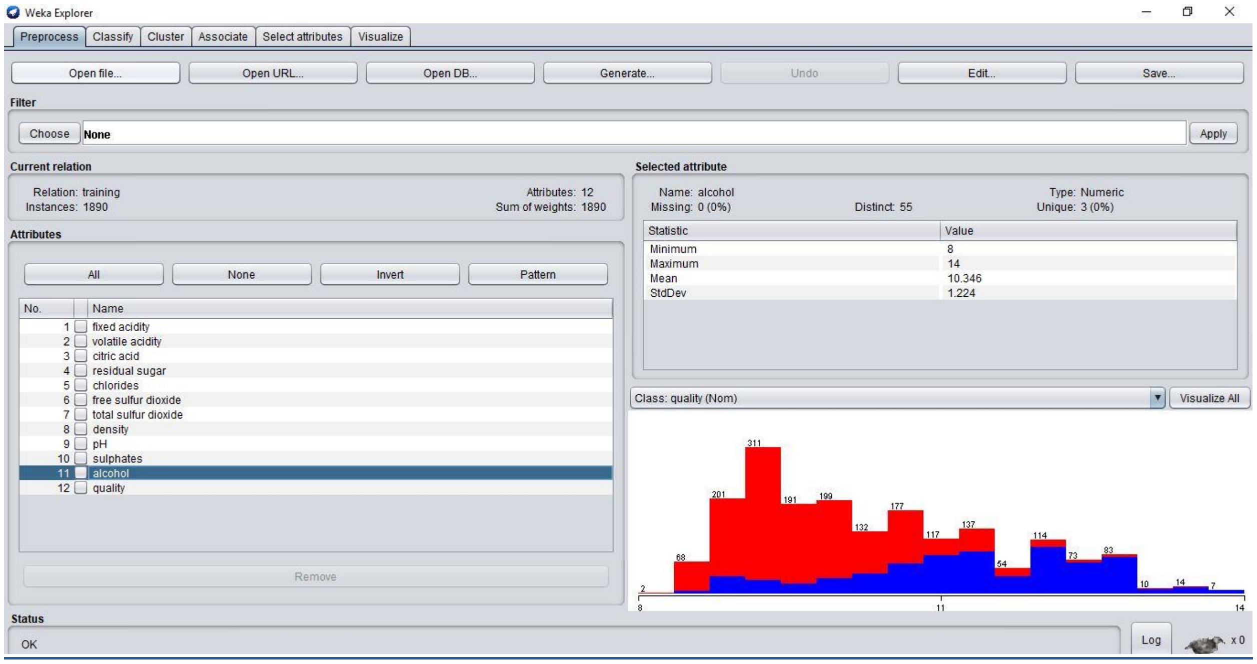
Figure 5: Histogram for the attribute “Volatile Acidity” showing the presence of outliers

As can be observed in the above figure, we can clearly demarcate the distribution of data and identify the possible outliers just by looking at the histogram. The points to the right appear as removed from the rest of the dataset highlighting them as possible outliers.

Based on the histogram, which attribute appears to be the most useful for classifying wine, and why?

Alcohol

When we look at the visualization of all the histograms, the Alcohol attribute stands out as the most useful for classifying wine. There is a clear demarcation in the progress of wine quality based on the alcohol content. We can observe how the alcohol values are distributed across the good and bad quality metric used in the classification model. This can be analyzed in the figure below.



Filter: None

Current relation

Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

Attributes

No.	Name
1	fixed acidity
2	volatile acidity
3	citric acid
4	residual sugar
5	chlorides
6	free sulfur dioxide
7	total sulfur dioxide
8	density
9	pH
10	sulphates
11	alcohol
12	quality

Selected attribute

Name: alcohol
Type: Numeric
Missing: 0 (0%)
Distinct: 55
Unique: 3 (0%)

Statistic	Value
Minimum	8
Maximum	14
Mean	10.346
StdDev	1.224

Class: quality (Nom)

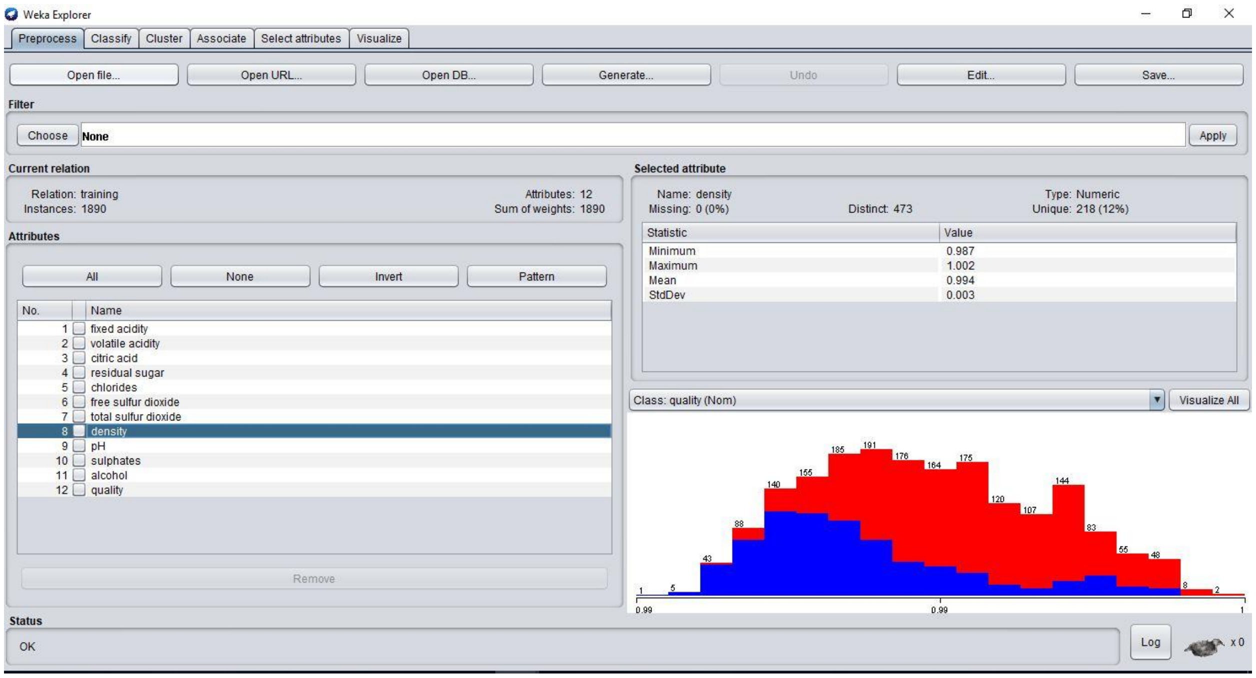
Status

OK

Figure 7: Showing the progress of good and bad quality wine based on alcohol content

We can observe as the value of the alcohol content increases, the color of the data changes from red to blue on the right. The red color distribution of data towards the left indicate that the lower the content of alcohol, the more likely the wine quality is bad. And the consistent blue color progress to the right implies that higher the alcohol content in the wine, the more good it becomes in its quality (based on the classifier chosen, blue is good quality and red is bad quality). We can observe that there are no outliers in this data making it viable for use in classification.

Another attribute that can also be considered useful is the density metric. As can be observed in figure below, this attribute shows a continuous distribution of data with no possible outliers. We can also observe the color progression from blue to red to the right indicating that more the density, the more likely the quality of wine is bad. However, it is not as consistent as observed in the attribute “alcohol”.


Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Generate... | Undo | Edit... | Save...

Filter

Choose None

Current relation

Relation: training
Instances: 1890
Attributes: 12
Sum of weights: 1890

Attributes

No.	Name
1	fixed acidity
2	volatile acidity
3	citric acid
4	residual sugar
5	chlorides
6	free sulfur dioxide
7	total sulfur dioxide
8	density
9	pH
10	sulphates
11	alcohol
12	quality

Selected attribute

Name: density
Missing: 0 (0%)
Distinct: 473
Type: Numeric
Unique: 218 (12%)

Statistic	Value
Minimum	0.987
Maximum	1.002
Mean	0.994
StdDev	0.003

Class: quality (Nom)

Status

OK

Figure 8: Showing the histogram for density

As for the above reasons, the alcohol attribute can be considered the most useful to classify wine based on the given dataset.

Task 1 – Part 2: Classifier Basics

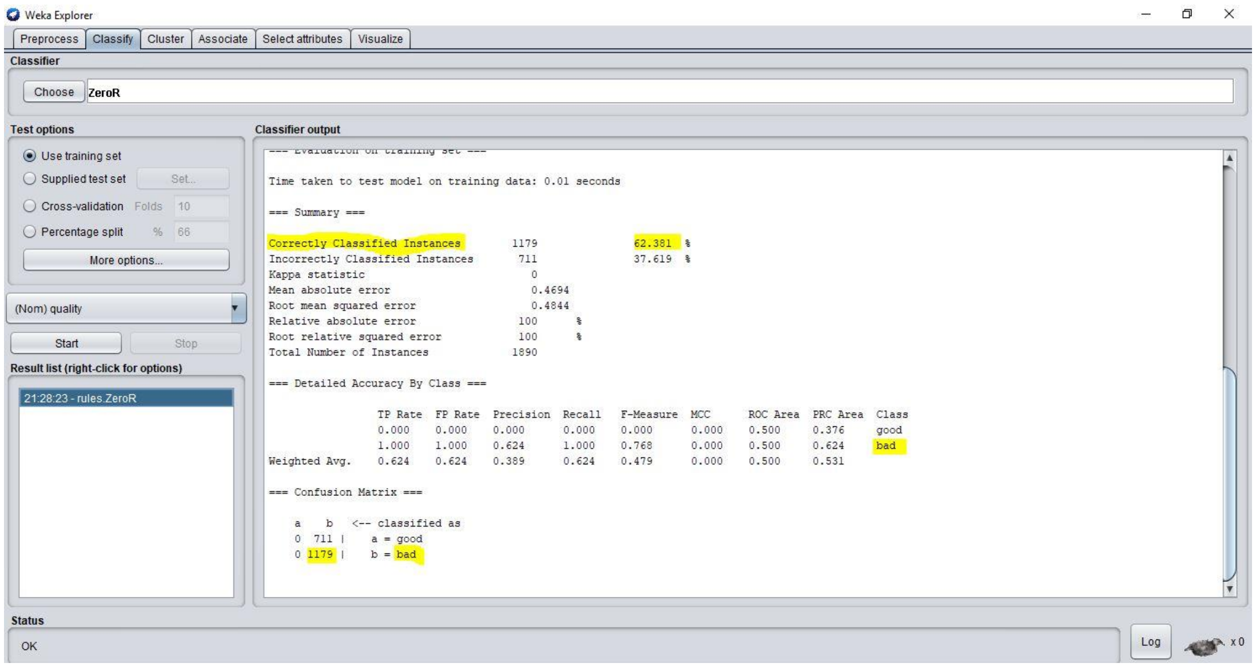
Question #2:

What is the *accuracy* - the percentage of correctly classified instances - achieved by *ZeroR* when you run it on the training set? Explain this number (what this number means ...). How is the accuracy of *ZeroR* a helpful baseline for interpreting the performance of other classifiers?

Answer:

What is the *accuracy* - the percentage of correctly classified instances - achieved by *ZeroR* when you run it on the training set? Explain this number (what this number means ...)

The accuracy achieved by ZeroR is 62.381%. It indicates the percentage of correctly classified instances as ‘bad’ in the training dataset. This refers to a total of 1179 as seen in the figure below.



ZeroR

Test options: (●) Use training set ( ) Supplied test set ( ) Cross-validation Folds 10 ( ) Percentage split % 66

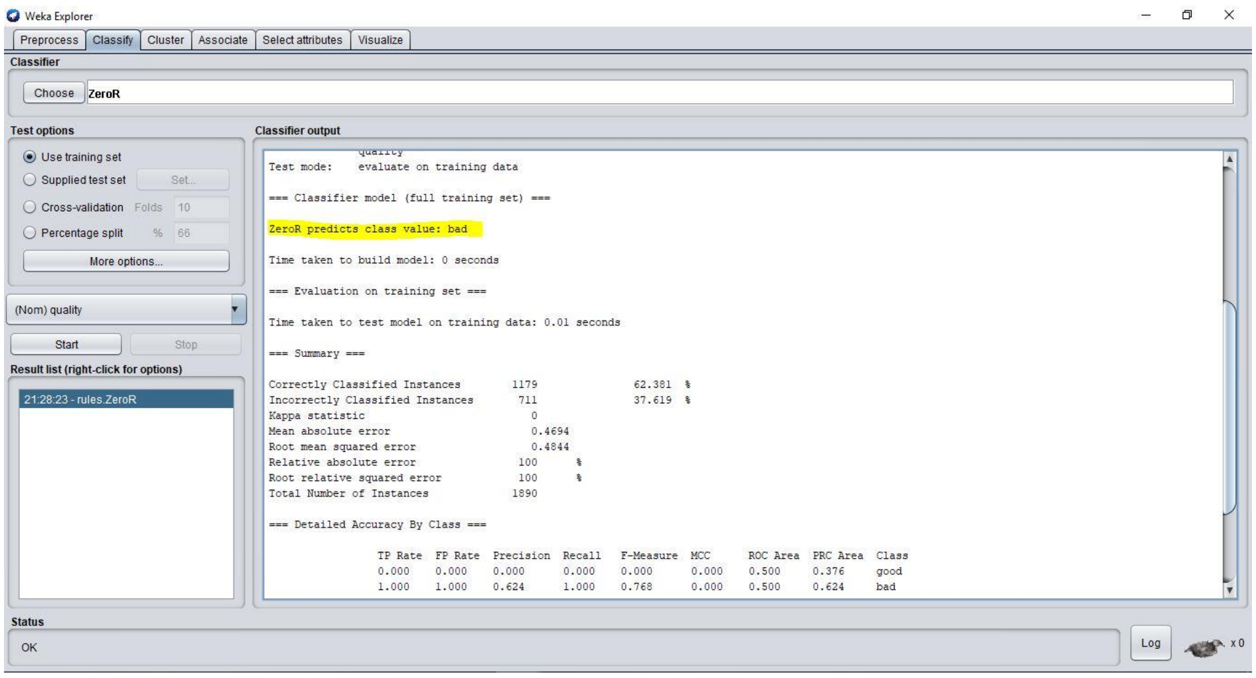
(Nom) quality

21:28:23 - rules.ZeroR

Status: OK

```
=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances        1179               62.381  %
Incorrectly Classified Instances       711               37.619  %
Kappa statistic                          0
Mean absolute error                      0.4694
Root mean squared error                  0.4844
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.376     good
                 1.000    1.000    0.624      1.000    0.768      0.000    0.500     0.624     bad
Weighted Avg.    0.624    0.624    0.389      0.624    0.479      0.000    0.500     0.531

=== Confusion Matrix ===

    a    b   <-- classified as
    0  711 |    a = good
    0 1179 |    b = bad
```

Figure 9: Showing the accuracy results for ZeroR along with the confusion matrix



ZeroR

Test options: (●) Use training set ( ) Supplied test set ( ) Cross-validation Folds 10 ( ) Percentage split % 66

(Nom) quality

21:28:23 - rules.ZeroR

Status: OK

```
              quality
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

ZeroR predicts class value: bad

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances        1179               62.381  %
Incorrectly Classified Instances       711               37.619  %
Kappa statistic                          0
Mean absolute error                      0.4694
Root mean squared error                  0.4844
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.376     good
                 1.000    1.000    0.624      1.000    0.768      0.000    0.500     0.624     bad
```

Figure 10: Showing the predicted class value by ZeroR

As indicated in the figures above, we can see that the class value predicted by the ZeroR algorithm is ‘bad’ (Figure 10). The ZeroR simply chooses the majority class in all the cases. We can observe the confusion matrix also and count the number of instances which are classified as ‘good’ or ‘bad’.

We can also observe the PRC area value to determine the accuracy of this algorithm. As compared to the ROC curve, which only gives us an idea of how the classifiers work in general, the Precision Recall Curve (PRC area) can be useful in determining how a classifier is behaving on one class. In this case, we are predicting to classify wine as ‘good’ or ‘bad’. So, we are not required to predict how many ‘good’ cases are correct, we want to predict correctly all the ‘bad’ cases and not miss any. This is observed by looking at the PRC area value and correlating them with the number of correctly classified instances and the confusion matrix.

Out of a total of 1890 instances, 1179 are correctly classified as ‘bad’ in the given training dataset. This accounts to a percentage of 62.381%.

How is the accuracy of *ZeroR* a helpful baseline for interpreting the performance of other classifiers?

The accuracy of the ZeroR can be a helpful baseline for other algorithms. Basically, as explained above, ZeroR is the simplest classification method which relies on the target and ignores all predictors. It simply predicts the majority category (class). Although there is no predictability power in ZeroR, the baseline performance can be used as a benchmark for other classification methods.

This baseline accuracy can be compared with other accuracy results that are obtained after running the several different classifier algorithms like J48 or Naïve Bayes. If these accuracy results are found to be less than the baseline accuracy of 62.381%, it would imply that the attributes in the dataset are not informative and may need further cleaning. This kind of method will stop us from blindly applying Weka on every dataset.

If another classifier is significantly better than ZeroR, then we can safely assume that it has accurately found some correlation between the attributes and classification.

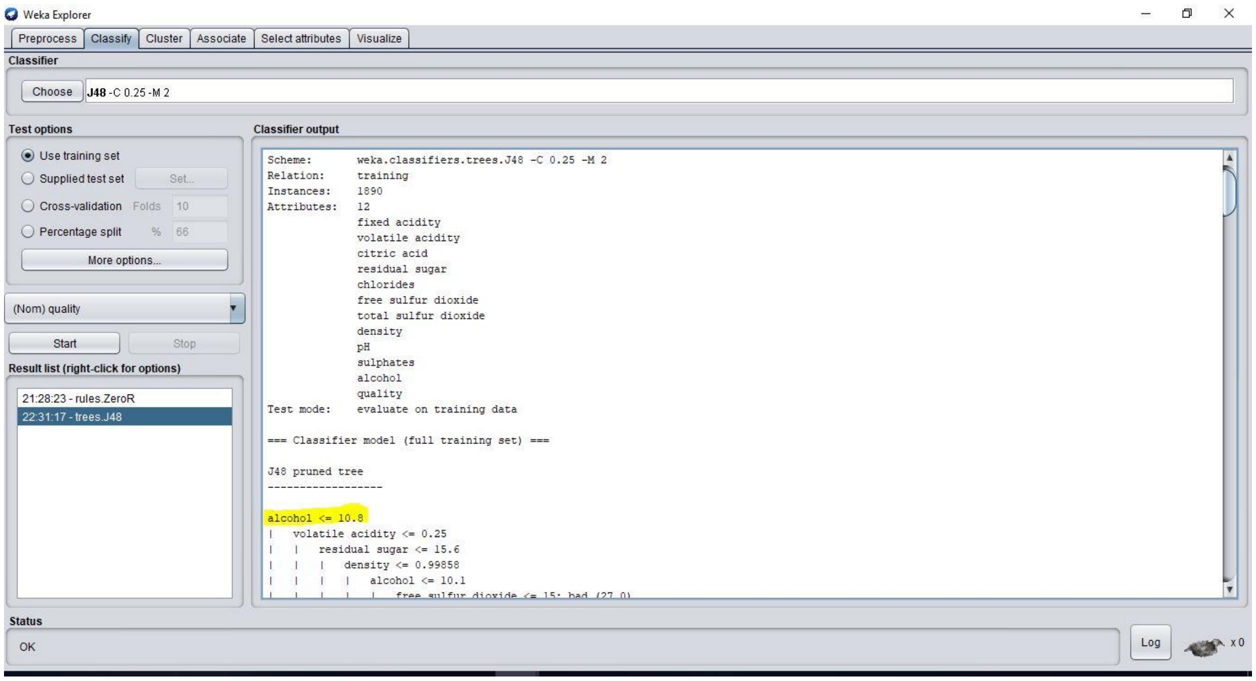
Question #3:

Using a decision tree Weka learned over the training set, what is the most informative single feature for this task, and what is its influence on wine quality? Does this match your answer from Question #1?

Answer:

Decision trees provide a set of rules that one can apply to a new dataset to predict the outcome. They are a tree like structure where each branch node represents a choice between alternatives and each leaf represents a classification. In the case of the chosen dataset, decision tree is the easiest way to visualize the relation between the class and the attributes.

We run the J48 algorithm and get the following result.



J48 -C 0.25 -M 2

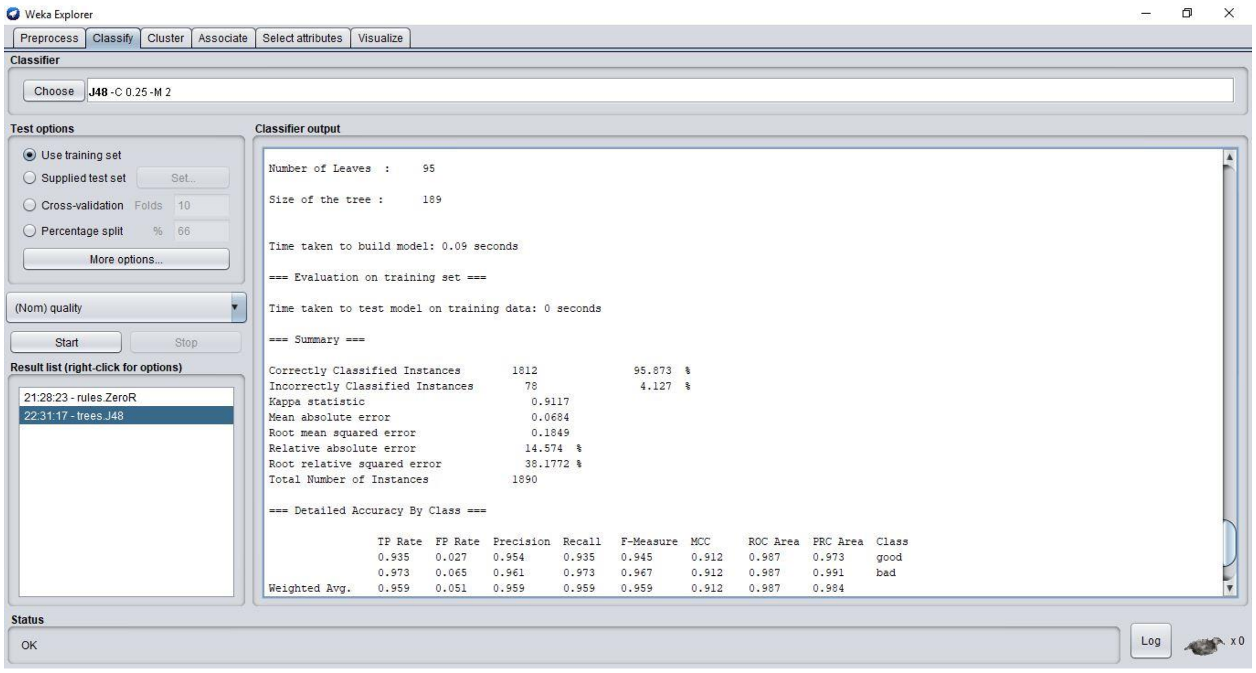
Use training set [✓]

(Nom) quality

Classifier output

```
Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     training
Instances:    1890
Attributes:   12
              fixed acidity
              volatile acidity
              citric acid
              residual sugar
              chlorides
              free sulfur dioxide
              total sulfur dioxide
              density
              pH
              sulphates
              alcohol
              quality
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
------------------

alcohol <= 10.8
|   volatile acidity <= 0.25
|   |   residual sugar <= 15.6
|   |   |   density <= 0.99858
|   |   |   |   alcohol <= 10.1
|   |   |   |   |   free sulfur dioxide <= 15: bad (27.0)
```

Figure11: Showing the root node 'alcohol' for the decision tree



J48 -C 0.25 -M 2

Use training set [✓]

(Nom) quality

Classifier output

```
Number of Leaves  :     95

Size of the tree :      189


Time taken to build model: 0.09 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances        1812               95.873  %
Incorrectly Classified Instances        78                4.127  %
Kappa statistic                          0.9117
Mean absolute error                      0.0684
Root mean squared error                  0.1849
Relative absolute error                 14.574  %
Root relative squared error             38.1772 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.935    0.027    0.954      0.935    0.945      0.912    0.987     0.973     good
                 0.973    0.065    0.961      0.973    0.967      0.912    0.987     0.991     bad
Weighted Avg.    0.959    0.051    0.959      0.959    0.959      0.912    0.987     0.984
```

Figure 12: Showing the number of leaves and the size of the tree

J48 is based on a top-down strategy, a recursive divide and conquer analysis. It selects an attribute to split at the root node and creates a branch for each possible attribute value. That in turn splits the instances into subsets one for each branch extended from the node. This process repeats recursively for each branch by selecting an attribute at each node and using only instances that reach the branch. Basically, it tries to choose the attribute with the 'purest' nodes that deliver the greatest information gain. This is calculated by measuring the information in bits and finding the entropy value of each attribute.

We now visualize the tree structure for this dataset.

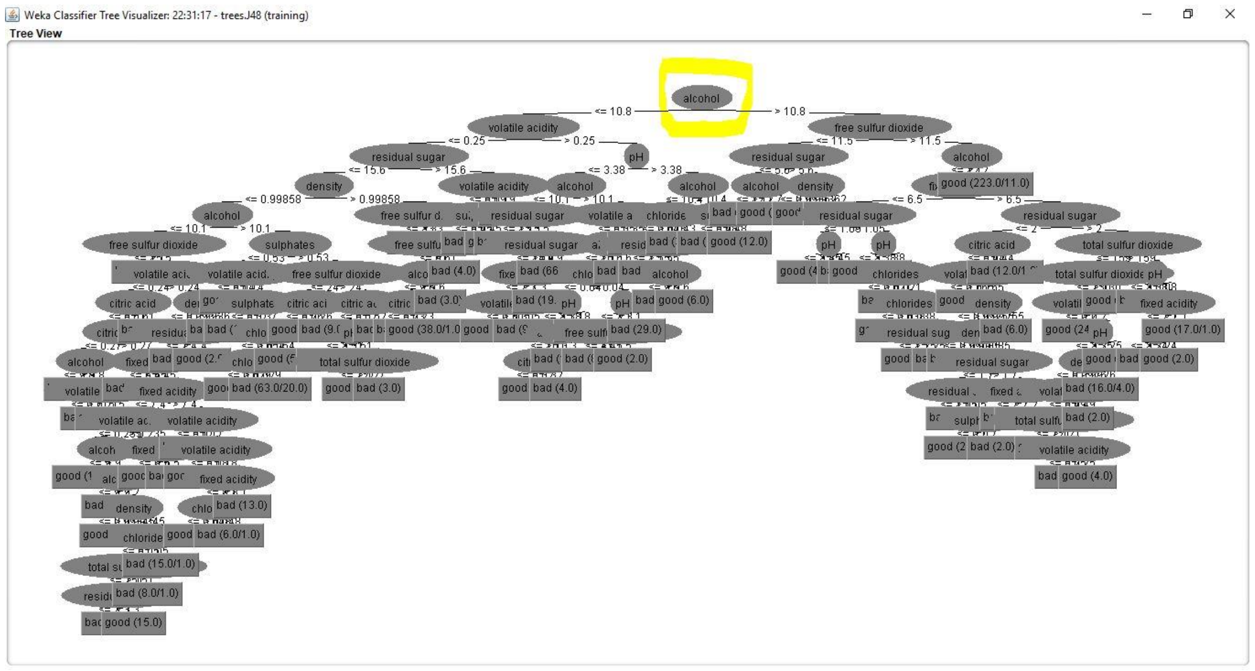


Figure 13: Showing the J48 tree visualization on the training dataset with the attribute 'alcohol' as its root node

As can be observed, from figures 11 and 13, the attribute 'alcohol' is chosen as the best attribute with the highest information gain. It lies at the root node of this decision tree making it the most informative single feature for this task. This is in accordance with the suggestion we made in 'answer 1', where we visually identified 'alcohol' as the best attribute to classify wine based on the histogram analysis.

By analyzing the data from the classifier output, we can clearly see that the value of the attribute 'alcohol' has a direct correlation with the quality of wine. The higher the alcohol value, the more likely the quality of wine is good. The lower the value, the more likely the quality of wine is bad.

Furthermore, the accuracy rate for this classifier is considerably high with an overall percentage of **95.873%** for the correctly classified instances. This include 665 instances classified as good and 1147 instances classified as bad in a total of 1812 correctly classified instances. This implies that the model is highly accurate in predicting the dataset and the chosen attribute 'alcohol' delivers the highest information gain.

We can safely conclude that our assumption of choosing ‘alcohol’ as the most useful attribute based on the histogram is in align with the results from the decision tree classifier.

Question #4:

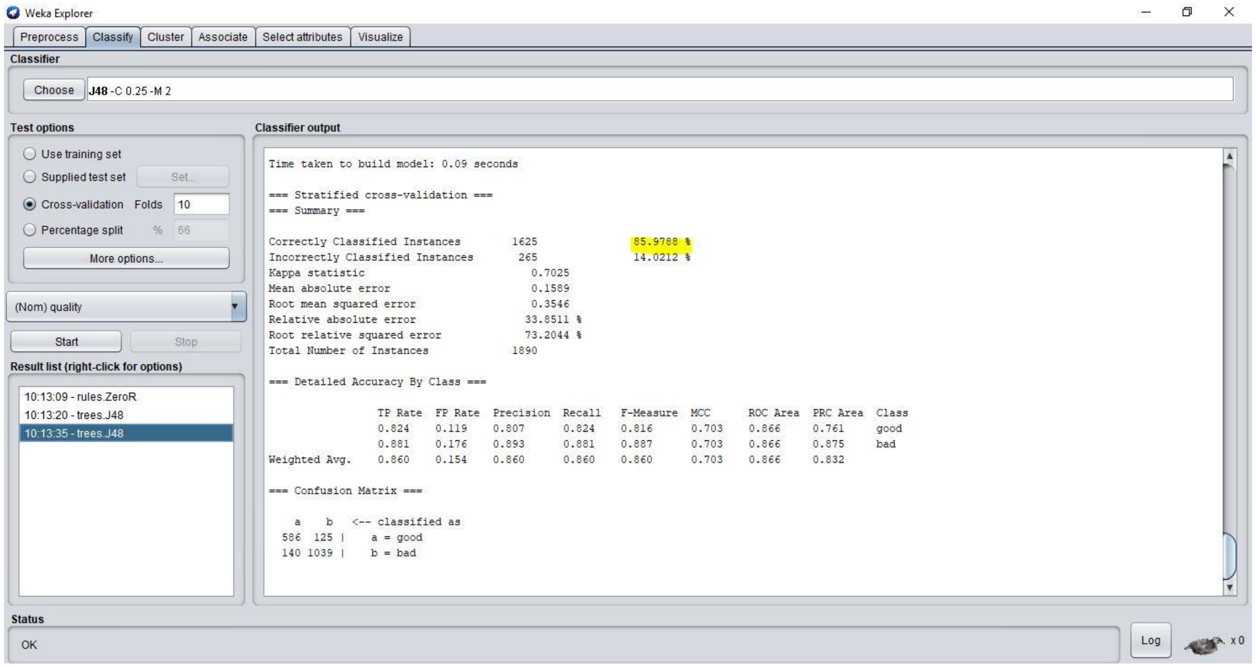
What is 10-fold cross-validation? What is the *main* reason for the difference between the percentage of Correctly Classified Instances when you used the entire training set directly versus when you ran 10-fold cross-validation on the training set? Why is cross-validation important?

Answer:

What is 10-fold cross-validation?

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

In 10-fold cross-validation, the original sample is randomly partitioned into 10 equal size subsamples. Of the 10 subsamples, a single subsample is retained as the validation data for testing the model, and the remaining 9 subsamples are used as training data. The cross-validation process is then repeated 10 times (the folds), with each of the 10 subsamples used exactly once as the validation data. The 10 results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.



J48 -C 0.25 -M 2

Cross-validation Folds 10

% 66

(Nom) quality

10:13:09 - rules.ZeroR
10:13:20 - trees.J48
10:13:35 - trees.J48

```
Time taken to build model: 0.09 seconds

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

Correctly Classified Instances        1625               85.9788 %
Incorrectly Classified Instances       265               14.0212 %
Kappa statistic                          0.7025
Mean absolute error                      0.1589
Root mean squared error                  0.3546
Relative absolute error                 33.8511 %
Root relative squared error             73.2044 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.824    0.119    0.807      0.824    0.816      0.703    0.866     0.761     good
                 0.881    0.176    0.893      0.881    0.887      0.703    0.866     0.875     bad
Weighted Avg.    0.860    0.154    0.860      0.860    0.860      0.703    0.866     0.832

=== Confusion Matrix ===

    a    b   <-- classified as
  586  125 |    a = good
  140 1039 |    b = bad
```

Figure 14: Showing the results of running the classifier with a 10-fold cross-validation

As we can observe, the accuracy of the correctly classified instances based on 10-fold cross validation is **85.9788%**. Even though this accuracy is high enough for predicting the model, it is comparatively lower than the accuracy we got while running the J48 classifier on the training dataset, which was **95.873%.**

What is the *main* reason for the difference between the percentage of Correctly Classified Instances when you used the entire training set directly versus when you ran 10-fold cross-validation on the training set? Why is cross-validation important?

Percentage of correctly classified instances using training dataset – 95.873%
Percentage of correctly classified instances using 10-fold cross validation – 85.9788%

Basically, cross validation is used to avoid overfitting and unfitting so that we can get an unbiased sense of model effectiveness. When a significant amount of data is at hand, we can set aside a few samples to evaluate the final model. In this case, our data consists of a total of 1890 instances which can be split into training and test sets to run the 10-fold cross-validation.

Cross-validation is a systematic way of doing repeated holdout that improves upon it by reducing the variance of the estimate. When we run our classifier on the training set and evaluate its performance, there is a certain amount of variance in that evaluation. We want to keep the variance of the estimate as low as possible for an accurate model. By using 10 cross-validation we reduce the variance of the final estimate. We can safely say that we get a better estimate of the final model using 10-fold cross-validation than running the classifier on the training set as a whole.

One of the reasons for the percentage difference is that when measuring accuracy with the training set, the decision tree algorithm is specifically tailored to perform well with the training set. This can give us an artificially high accuracy. By splitting the data into a training set and a test set, we are more closely approximating the performance of the decision tree on a set that it was not specifically tailored for. This makes 10-fold cross-validation very important, as it can give us a much better idea of how our decision tree performs in general.

Question #5:

What is the "command-line" for the model you are submitting? For example, *"J48 -C 0.25 -M 2"*. What is the reported accuracy for your model using 10-fold cross-validation?

Command-line for the model – “J48 -U -M 2 -A”

This command line refers to the several parameters set for the unpruned decision tree classifier we are using.

U stands for unpruned.

M = 2 indicates the number of instances per leaf which is 2, minNumObj

-A indicates the useLaplace parameter to smoothen out the counts of leaves using laplace method.

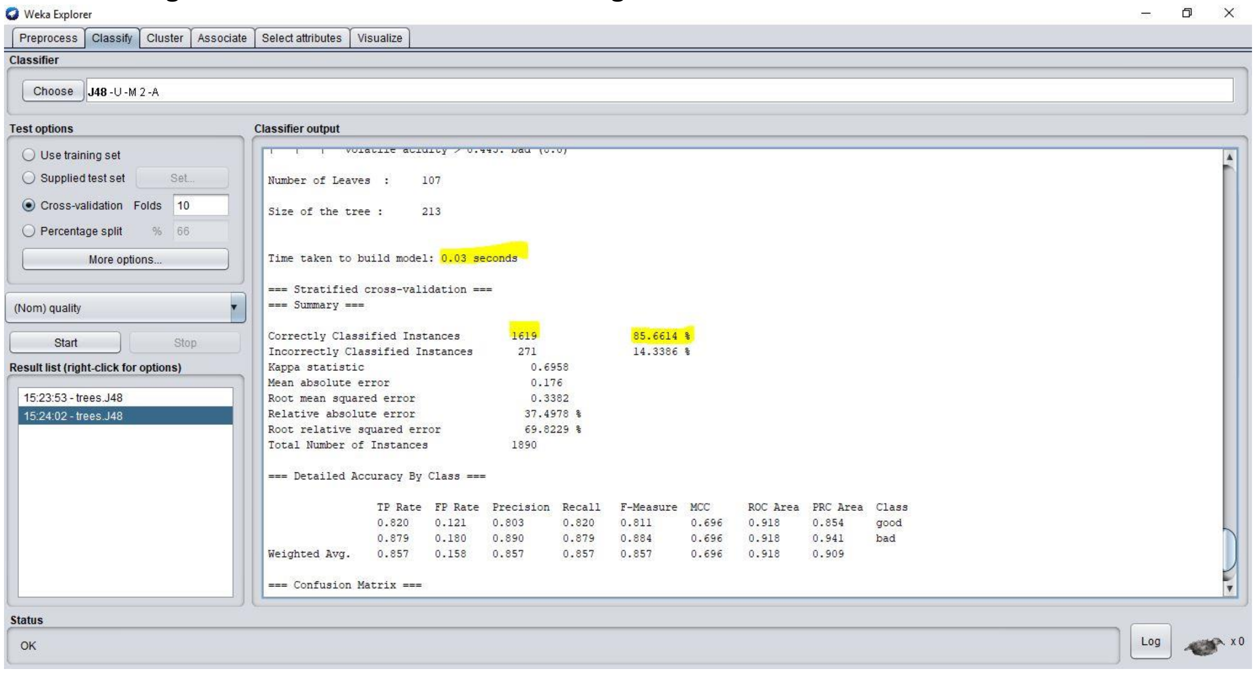
All the other parameters are set to default in Weka.

10-fold Cross Validation

Cross-validation can be used to improve upon the repeated holdout method which we just used with the random-number seeds. Cross validation is a systematic way of using repeated holdout that improves upon it by reducing the variance of the estimate. Stratified cross validation reduces the variance even further. Weka does Stratified cross validation by default. With 10-fold cross validation, Weka invokes the learning algorithm 11 times, one for each fold of the cross validation and then a final time on the entire dataset.

A practical rule of thumb is, if we have a large dataset, we can use percentage split and evaluate it just once. Otherwise, if we don't have too much, we should use stratified 10-fold cross validation. Large dataset usually depends on the number of classes in the dataset. The chosen dataset is not as large and cross-validation can be a reliable way to evaluate it.

The following results are obtained after running 10-fold cross validation on the chosen model.



J48 -U -M 2 -A

Cross-validation Folds 10

Percentage split % 66

(Nom) quality

```
Number of Leaves  :     107

Size of the tree :      213


Time taken to build model: 0.03 seconds

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

Correctly Classified Instances        1619               85.6614 %
Incorrectly Classified Instances       271               14.3386 %
Kappa statistic                          0.6958
Mean absolute error                      0.176
Root mean squared error                  0.3382
Relative absolute error                 37.4978 %
Root relative squared error             69.8229 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.820    0.121    0.803      0.820    0.811      0.696    0.918     0.854     good
                 0.879    0.180    0.890      0.879    0.884      0.696    0.918     0.941     bad
Weighted Avg.    0.857    0.158    0.857      0.857    0.857      0.696    0.918     0.909

=== Confusion Matrix ===
```

Figure 15: Showing the results of 10-fold cross validation

10-Fold Cross validation results:

```
Number of Leaves:       107
Size of the tree:       213
Time taken to build model: 0.03 seconds
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances              1619          85.6614 %
Incorrectly Classified Instances            271           14.3386 %
Kappa statistic                             0.6958
Mean absolute error                         0.176
Root mean squared error                     0.3382
Relative absolute error                     37.4978 %
Root relative squared error                 69.8229 %
Total Number of Instances                   1890
```

As it can be observed, the model performs well with the given parameters when run using the 10-fold cross validation. Weka runs the stratified cross validation by default. The accuracy of the correctly classified instances is high (85.6614%) with 1619 correctly classified instances out of a total of 1890.The time taken to build the model is also significantly faster (0.03) indicating a good balance between performance and speed.

It can also be noted that the relative absolute error value (37.4978%) is almost double to what we obtained while running the algorithm on the training set. This indicates that the data may be prone to more error while calculating the end results using this setting. However, the percentage of error can be considered normal based on the increased size of the tree and the number of leaves

As explained above, cross validation can be used to reduce the variance of data. By using variance, we can measure how far a set of random numbers are spread out from their average value. Reducing the variance using cross validation can improve the readability of the data for future models.

For the aforesaid reasons, we can assume that 10-fold cross validation is an optimum way of predicting the accuracy of the data using the chosen model.

The accuracy of the chosen model using 10-fold cross validation is 85.6614%.

Question #6:

In a few sentences, describe how you chose the model you are submitting. Be sure to mention your validation strategy and whether you tried varying any of the model parameters.

Several evaluation methods were used to determine the model for classification. The following methods acted as a guideline in predicting the final model.

Evaluation methods:
• Predictive (Classification) accuracy: This refers to the ability of the model to correctly predict the class label of new or previously unseen data.
• Accuracy = % of testing set examples correctly classified by the classifier
• Speed: This refers to the computation costs involved in generating and using the model
• Robustness: This is the ability of the model to make correct predictions given noisy data or data with missing values
•Scalability: This refers to the ability to construct the model efficiently given large amount of data
• Interpretability: This refers to the level of understanding and insight that is provided by the model
• Simplicity: • decision tree size • rule compactness • Domain-dependent quality indicators

The given dataset is clean without any missing values. The two significant evaluation criteria chosen includes the classification accuracy of the correctly classified instances along with the time taken to build the model.

Identifying the classifier:

The different classifiers used on the training set and their results are displayed in a table format below.

Algorithm	**Accuracy (correctly classified instances)**	**Time taken to build data**	**Time taken to test data**
ZeroR	62.381%	0 seconds	0.63 seconds
MultilayerPerceptron	89.8413%	1.62 seconds	0.06 seconds
J48	95.873%	0.02 seconds	0 seconds
Naïve-Bayes	78.9418%	0.2 seconds	0.03 seconds
Lazy-IBK (k=5)	89.418%	0.01 seconds	0.23 seconds

We can summarize the results of the table as below

- The J48 classifier outperforms the other algorithms in relation to accuracy and performance.
- Even though the instance-based IBK delivers a highly accurate result, the time taken to test the model is also high.

- The MultilayerPerceptron classifier also provides a high accuracy of 89.8413%, but the time taken to build the model is significantly high. \
- The probability based Naïve-Bayes classifier provided a decent accuracy, however the data representation using the decision tree algorithm is significantly easy in predicting the class simply based on the numerical value of the given attributes.

For the aforesaid reasons, J48 was chosen as the best classifier to predict the model with a high accuracy.

Setting up the different parameters for J48 classifier:

We now use several different parameters to find the most accurate and time efficient combination for the decision tree algorithm.

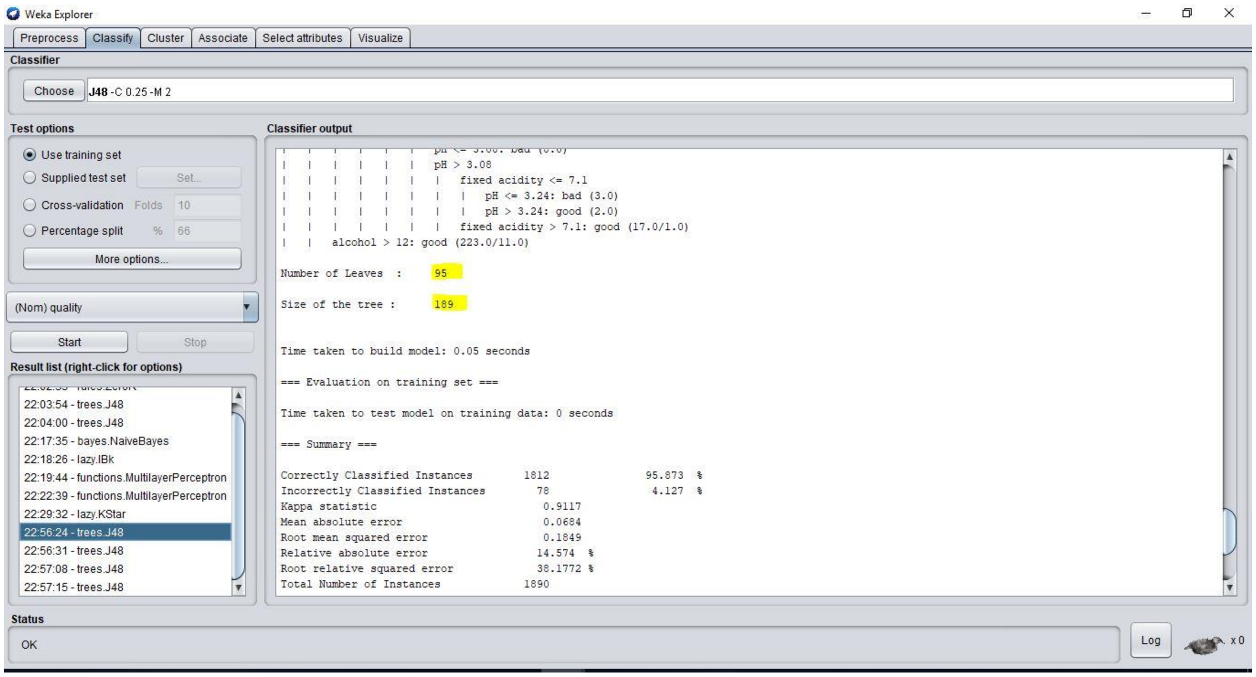
We ran the classifier with several different combinations which include 10-fold cross validation, percentage split of 90 and 66, along with the training data. The results are shown in the table below.

Algorithm Test Option	**Accuracy (correctly classified instances)**	**Time taken to build data**	**Time taken to test data**
Training set	95.873%	0.02 seconds	0 seconds
Cross Validation (10-fold)	85.9788%	0.04 seconds	0.02 seconds
Cross Validation (5-Fold)	86.8254%	0.04 seconds	0.03 seconds
Percentage Split (90%)	86.7725%	0.04 seconds	0 seconds
Percentage Split (66%)	83.9813%	0.04 seconds	0 seconds

By comparing the results above, we can clearly see that the classifier performs well with all the chosen test options. However, the basic training set is delivering the most accurate model with a higher performance rate. This training set is the default pruned tree structure as run by Weka. Changing the value of the random number seed from 1 to 10 using 10-fold cross validation did not seem to have much effect on the accuracy of the model.

Pruned and Unpruned Tree:

We run the J48 algorithm on the training set in both pruned and unpruned mode as shown in the figure below.

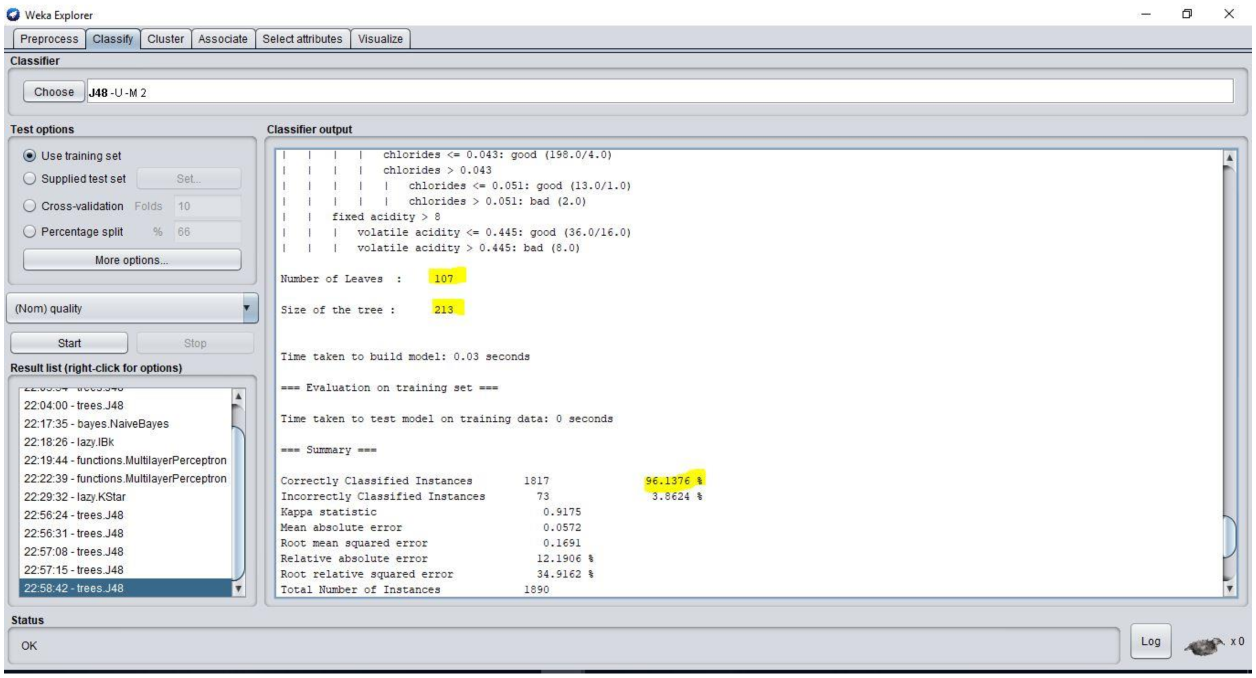

Classifier: J48 -C 0.25 -M 2

Test options: Use training set; (Nom) quality

```
|   |   |   |   |   |   |   pH > 3.08
|   |   |   |   |   |   |   |   fixed acidity <= 7.1
|   |   |   |   |   |   |   |   |   pH <= 3.24: bad (3.0)
|   |   |   |   |   |   |   |   |   pH > 3.24: good (2.0)
|   |   |   |   |   |   |   |   fixed acidity > 7.1: good (17.0/1.0)
|   |   alcohol > 12: good (223.0/11.0)

Number of Leaves  :     95

Size of the tree :      189


Time taken to build model: 0.05 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances        1812               95.873  %
Incorrectly Classified Instances        78                4.127  %
Kappa statistic                          0.9117
Mean absolute error                      0.0684
Root mean squared error                  0.1849
Relative absolute error                 14.574  %
Root relative squared error             38.1772 %
Total Number of Instances             1890
```

Status: OK

Figure 16: Showing the Pruned tree results for J48


Classifier: J48 -U -M 2

Test options: Use training set; (Nom) quality

```
|   |   |   |   chlorides <= 0.043: good (198.0/4.0)
|   |   |   |   chlorides > 0.043
|   |   |   |   |   chlorides <= 0.051: good (13.0/1.0)
|   |   |   |   |   chlorides > 0.051: bad (2.0)
|   |   fixed acidity > 8
|   |   |   volatile acidity <= 0.445: good (36.0/16.0)
|   |   |   volatile acidity > 0.445: bad (8.0)

Number of Leaves  :     107

Size of the tree :      213


Time taken to build model: 0.03 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances        1817               96.1376 %
Incorrectly Classified Instances        73                3.8624 %
Kappa statistic                          0.9175
Mean absolute error                      0.0572
Root mean squared error                  0.1691
Relative absolute error                 12.1906 %
Root relative squared error             34.9162 %
Total Number of Instances             1890
```

Status: OK

Figure 17: Showing the Unpruned Tree results for J48

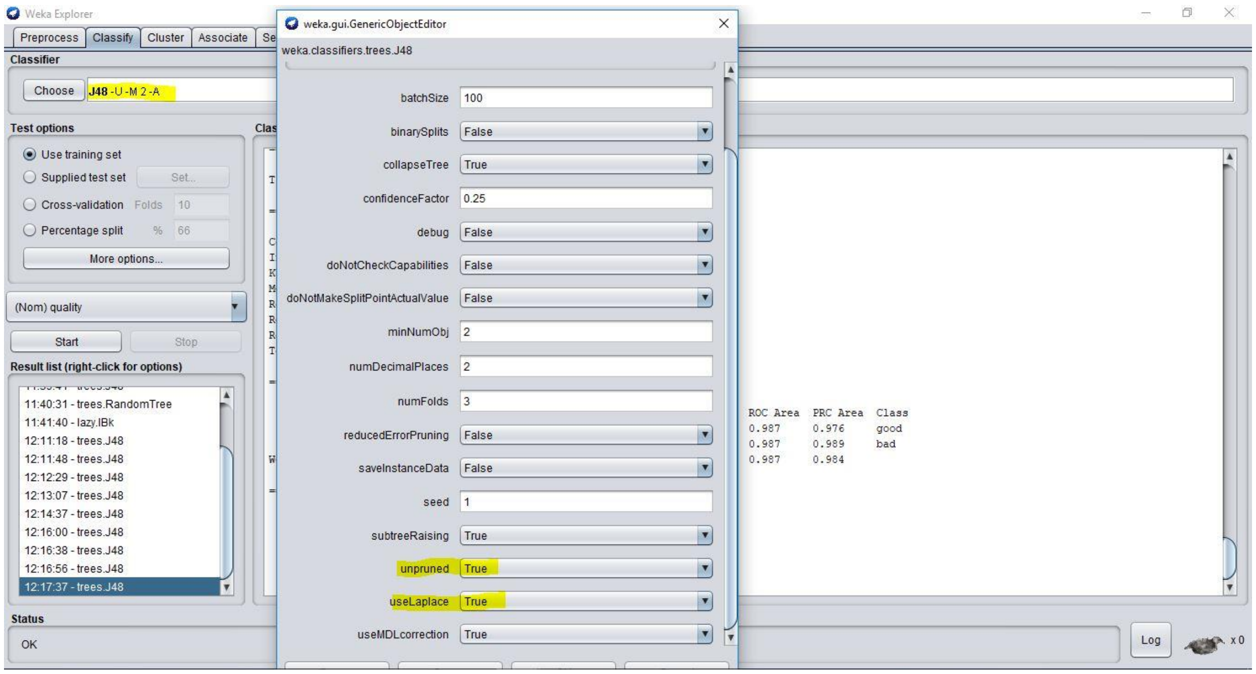
As can be observed, the Unpruned tree has more number of leaves and branches (107 and 213 respectively) with a marginally higher accuracy than the Pruned tree (with 95 leaves and 189 branches). The time taken to build the model is almost the same. More number of leaves may make it slightly complicated to visualize the tree structure using the unpruned model. However, the unpruned tree structure has only a 10% increase in its size as compared to the pruned tree, which can be considered minimal as the total number of instances in the given dataset is also not very high (1890 in total). Even the accuracy of the Pruned tree (95.873%) is almost the same as the Unpruned one (96.1376%).

Considering the aforesaid arguments, we can safely assume the Unpruned tree structure can be chosen as a suitable model for our classification purposes.

(The Pruned and Unpruned mode is achieved by setting the parameter under Unpruned in the command line to ‘False’ or ‘True’)

Parameters chosen:

We now try to tweak a few more parameters within the command line that might help us in getting even better results.



J48 -U -M 2 -A

11:40:31 - trees.RandomTree
11:41:40 - lazy.IBk
12:11:18 - trees.J48
12:11:48 - trees.J48
12:12:29 - trees.J48
12:13:07 - trees.J48
12:14:37 - trees.J48
12:16:00 - trees.J48
12:16:38 - trees.J48
12:16:56 - trees.J48
12:17:37 - trees.J48

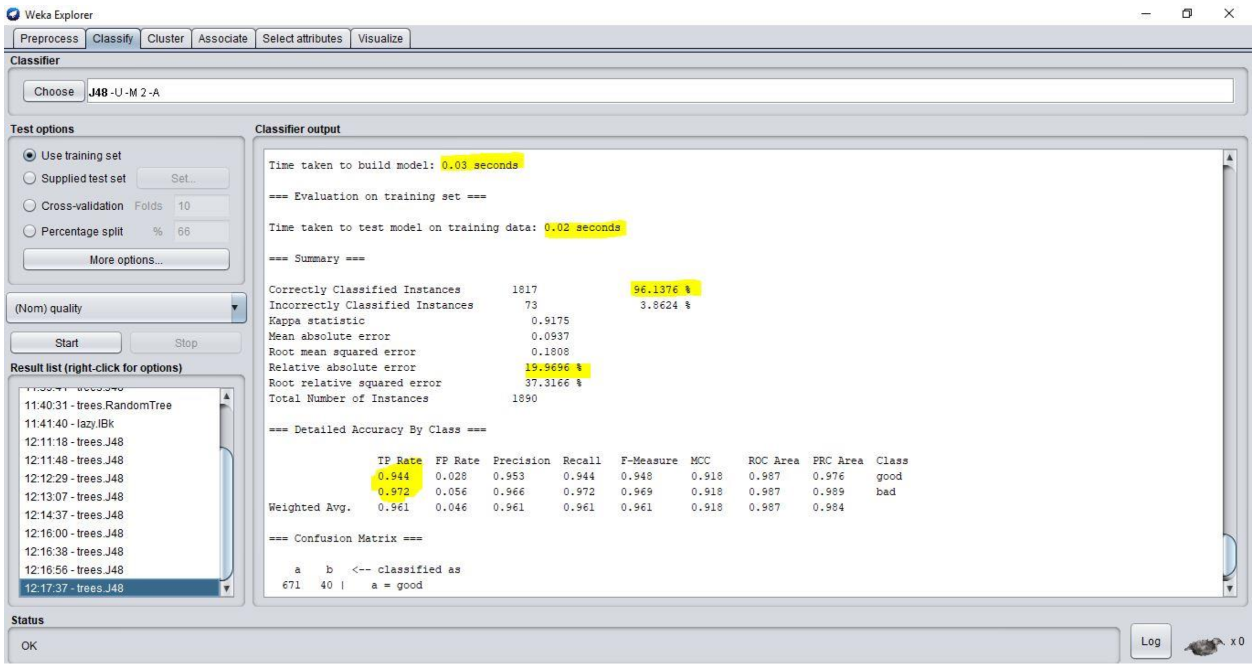
weka.classifiers.trees.J48

batchSize	100
binarySplits	False
collapseTree	True
confidenceFactor	0.25
debug	False
doNotCheckCapabilities	False
doNotMakeSplitPointActualValue	False
minNumObj	2
numDecimalPlaces	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	True
useLaplace	True
useMDLcorrection	True

```
ROC Area  PRC Area  Class
0.987     0.976     good
0.987     0.989     bad
0.987     0.984
```

Figure 18: Showing the parameter settings for the Unpruned J48 classifier

The basic parameters are set to default. We changed the useLaplace option to True. This will smoothen the counts at the leaves using Laplace. These settings gave a slightly higher accuracy of 96.1376%. The time taken to build the model was also very fast as shown in the figure below.



J48 -U -M 2 -A

(●) Use training set
( ) Supplied test set
( ) Cross-validation Folds 10
( ) Percentage split % 66

(Nom) quality

```
11:40:31 - trees.RandomTree
11:41:40 - lazy.IBk
12:11:18 - trees.J48
12:11:48 - trees.J48
12:12:29 - trees.J48
12:13:07 - trees.J48
12:14:37 - trees.J48
12:16:00 - trees.J48
12:16:38 - trees.J48
12:16:56 - trees.J48
12:17:37 - trees.J48
```

```
Time taken to build model: 0.03 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.02 seconds

=== Summary ===

Correctly Classified Instances        1817               96.1376 %
Incorrectly Classified Instances        73                3.8624 %
Kappa statistic                          0.9175
Mean absolute error                      0.0937
Root mean squared error                  0.1808
Relative absolute error                 19.9696 %
Root relative squared error             37.3166 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.944    0.028    0.953      0.944    0.948      0.918    0.987     0.976     good
                 0.972    0.056    0.966      0.972    0.969      0.918    0.987     0.989     bad
Weighted Avg.    0.961    0.046    0.961      0.961    0.961      0.918    0.987     0.984

=== Confusion Matrix ===

    a    b   <-- classified as
  671   40 |    a = good
```

Figure 19: Showing higher accuracy results with the set parameters

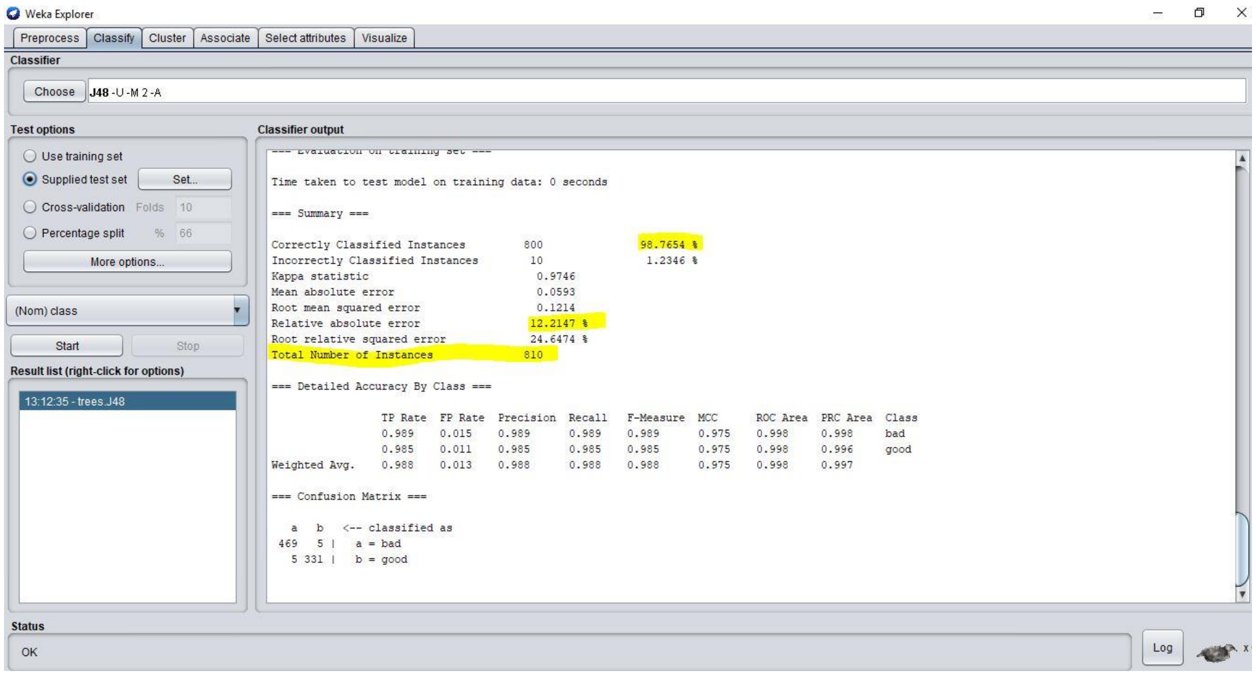
The Relative absolute error is also at 19.9696% which is significantly lower with a True Positive rate of 0.944 in predicting the class attribute value as ‘good’ and 0.972 in predicting the class value as ‘bad’.

Changing other parameters like minNumOj which define the minimum number of instances per leaf increased the accuracy closer to 98% but the size of the tree became much bigger to 275 with 138 leaves. This was not considered a viable compromise for the increase in the tree size and the accuracy of the model. Hence, it was discarded.

After going through and analyzing a series of various combinations, the above set parameters (as detailed in figure 17) were chosen to create the final predictive model.

Using the classifier on the predicted test dataset:

The classifier was used on the predicted test dataset that was obtained. The results showed that the model was very accurate in predicting the test dataset as well. It showed an accuracy rate of 98.7654% with a very low relative absolute error rate of 12.2147% as detailed in the figure below.



J48 -U -M 2 -A

```
=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances         800               98.7654 %
Incorrectly Classified Instances        10                1.2346 %
Kappa statistic                          0.9746
Mean absolute error                      0.0593
Root mean squared error                  0.1214
Relative absolute error                 12.2147 %
Root relative squared error             24.6474 %
Total Number of Instances              810

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.989    0.015    0.989      0.989    0.989      0.975    0.998     0.998     bad
                 0.985    0.011    0.985      0.985    0.985      0.975    0.998     0.996     good
Weighted Avg.    0.988    0.013    0.988      0.988    0.988      0.975    0.998     0.997

=== Confusion Matrix ===

   a   b   <-- classified as
 469   5 |   a = bad
   5 331 |   b = good
```

Figure 20: Showing the accuracy results for the predicted test dataset with 810 instances

*<u>*Note – This file is provided as a separate A1-Udayabhaskarreddy-Malkannagari.csv in the assignment folder.</u>*

Question #7:

For your selected classifier model, investigate what kind of parameters are available to be set in WEKA and briefly summarize of the role of each parameter.
Throughout the various configuration testing, summarize your findings about the effect (on the result of testing the model using test data) of three different parameters of your choice.

Under the J48 classifier, we can choose a set of various parameters to tweak the final predictive model for better results.

We now look at the list of different parameters available in Weka for this classifier.

Available Parameters

<u>minNumObj:</u>

The minNumObj is set to the default 2 as it represents the minimum number of instances per leaf. This might be critical in case of the test dataset as altering its value may affect the predictive accuracy of the test data.

confidenceFactor:

The confidence factor is used for pruning, the smaller the value of the confidence factor, the more pruned the tree is. By default, it is set to 0.25. As we are running an unpruned tree, the confidence factor does not affect the end result of classification. If we opt to run the classifier as a pruned tree, we can tweak this setting for a better end result.

useLaplace:

It determines whether counts at leaves are smoothed based on Laplace. The useLaplace is set to True in order to smooth the counts of leaves based on Laplace.

minNumObj:

The minimum number of instances per leaf. This is set to 2 by default.

useMDLcorrection:

It determines whether MDL correction is used when finding splits on numeric attributes. Set to True by default.

seed:

The seed used for randomizing the data when reduced-error pruning is used. Deafault is set to 1

reducedErrorPruning:

Determines whether reduced-error pruning is used instead of C.4.5 pruning.

numFolds:

Determines the amount of data used for reduced-error pruning. One-fold is used for pruning, the rest for growing the tree. Default it is set to 3.

numDecimalPlaces:

The number of decimal places to be used for the output of numbers in the model.

batchSize:

The preferred number of instances to process if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.

Testing the model using three different parameter settings:

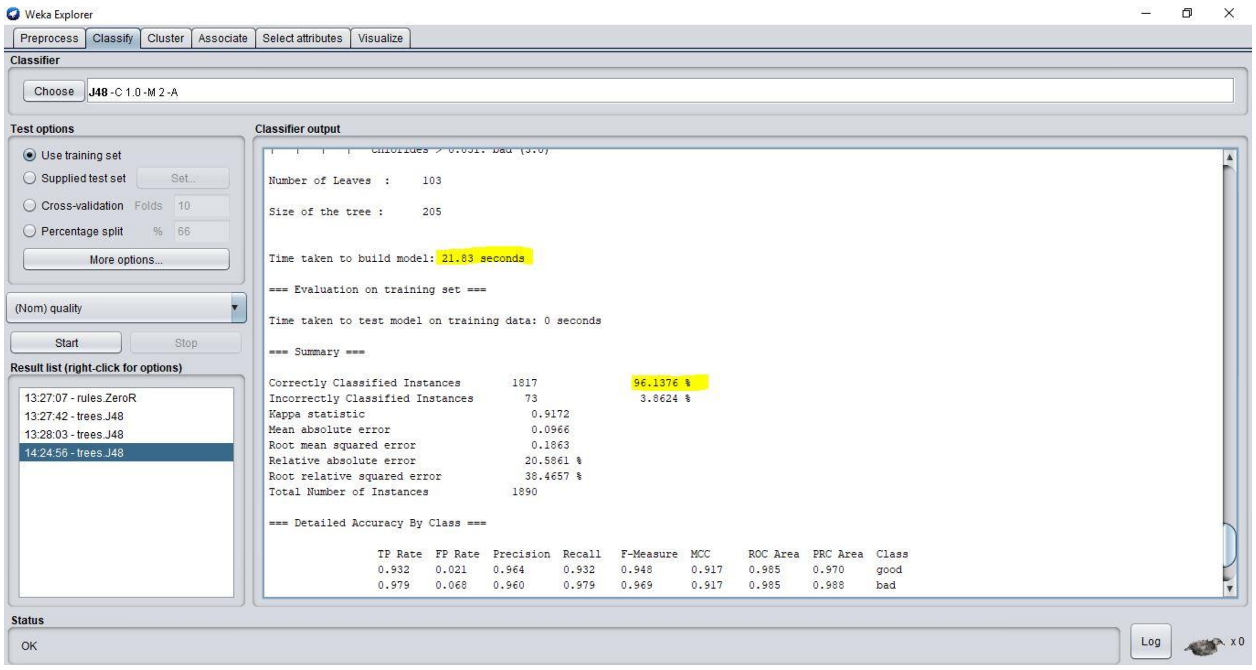
1- Changing confidenceFactor to 1 and using the pruned tree mode:

We now change the confidence factor setting to 1 and see its effect on the classification results. But in order to do that, we have to set the Unpruned option to False, because confidence factor is used for pruning the tree. The lower the value of the confidence factor the higher the pruning.

All the other options were left as it based on the chosen classifier model.

As we can see from the figure below, the accuracy of the classifier is the same as the chosen one. However, the time taken to build the model was considerably larger, 21.83 seconds. This does not reflect an optimum performance using the chosen setting. The confidenceFactor was reverted back to the default 0.25 and the algorithm was run again in the pruned mode. This time, the time taken to build the model was very fast, 0.09 seconds. Similar results were observed when the model was tested using the test data.

Hence, we can safely assume that lower values of confidence factor yield a faster model for the given dataset.



Classifier: J48 -C 1.0 -M 2 -A

Test options: [✓ Use training set] Supplied test set; Cross-validation Folds 10; Percentage split % 66

(Nom) quality

Result list:
13:27:07 - rules.ZeroR
13:27:42 - trees.J48
13:28:03 - trees.J48
14:24:56 - trees.J48

```
Number of Leaves  :     103

Size of the tree :      205


Time taken to build model: 21.83 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances        1817               96.1376 %
Incorrectly Classified Instances        73                3.8624 %
Kappa statistic                          0.9172
Mean absolute error                      0.0966
Root mean squared error                  0.1863
Relative absolute error                 20.5861 %
Root relative squared error             38.4657 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.932    0.021    0.964      0.932    0.948      0.917    0.985     0.970     good
                 0.979    0.068    0.960      0.979    0.969      0.917    0.985     0.988     bad
```

Status: OK

Figure 21: Showing the results of the classifier with confidenceFactor set to 1

Training data:

confidenceFactor	**Accuracy**	**Time taken to build the model**
1	96.1376%	21.83 seconds
0.25	95. 873%	0.09 seconds

Test Data:

confidenceFactor	**Accuracy**	**Time taken to build the model**
1	98.7654%	8.42 seconds
0.25	97. 873%	0.09 seconds

2- Changing the minNumObj (minimum number of instances per leaf)

We now try to change the setting minNumObj on the classifier model we chose. This represents the minimum number of instances per leaf. By default, it is set to 2. We change it to 1 and 3, and compare the results.

Training Data:

minNumObj	**Accuracy**	**Number of leaves**	**Size of the tree**
2	96.1376%	107	213
1	97.4603%	138	275
3	95.3439%	100	199

Test Data:

minNumObj	**Accuracy**	**Number of leaves**	**Size of the tree**
2	98.7654%	50	99
1	98.8765%	60	119
3	98.1481%	45	89

We can clearly see that the change in the number of instances per leaf is affecting the size of the tree structure. When the minNumObj is set to 1, we are getting a higher accuracy but at the cost of a larger tree. When it is set to 3, we are getting a smaller tree with a slightly lower accuracy.

The same can be observed when we used the classifier on the test dataset. The size of the tree is larger when the minimum number of instances per leaf is set to 1.

By the above results, we can conclude that setting the instances per leaf to 1 will give us a bigger decision tree which will visually difficult to comprehend as compared to the default setting.

3- **Changing the reducedErrorPruning and numFolds option:**

We now set the reducedErrorPruning option to True and increase the number of numFolds to 5 and the compare the results. numFolds determines the amount of data used for error pruning. 1 fold is used for pruning and the rest to grow the tree.

reducedErrorPruning determines the method of pruning used. By setting it to True, we use reduced error pruning instead of the C 4.5 pruning.

Test Data:

numFolds	**reducedErrorPruning**	**Accuracy**	**Number of leaves**	**Size of the tree**
5	True	91.1111%	14	27
3	False	98.1481%	43	85

As we can observe, the tree size becomes much smaller with the increase in the number of folds. This decision tree will be very easy to visualize and comprehend as shown in the figure below.

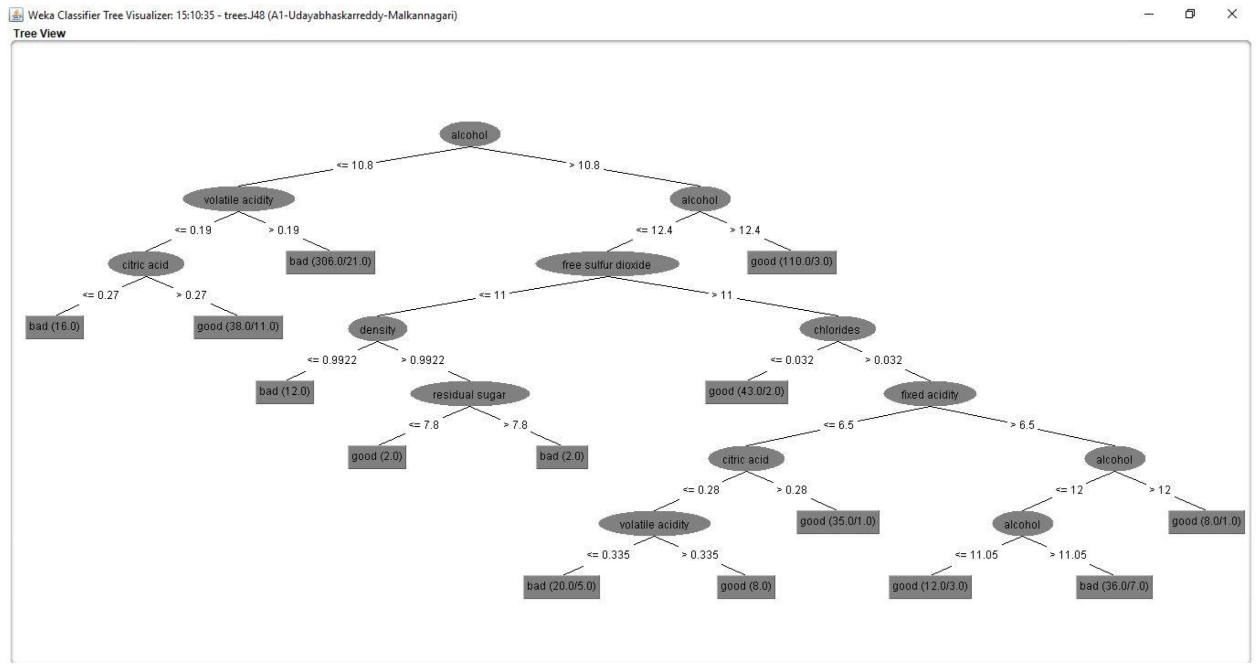


Figure 22: Showing the decision tree visualization of the test data

Even the accuracy of correctly classified instances is also high making it a competent parameter setting for predicting future models.

Task#2 – Try Another Data Set

Question #8:

Briefly explain what strategy you used to obtain the Classifiers A and B that performed well on one of the car or wine data sets, and not the other.

Dataset 1 – Wine data
Dataset 2 – Car evaluation data

Car Evaluation Dataset
Number of Instances: 1190
Number of Attributes: 6
Attribute Values:

buying	v-high, high, med, low
maint	v-high, high, med, low
doors	2, 3, 4, 5-more
persons	2, 4, more

lug_boot	small, med, big
safety	low, med, high
Class	unacc, acc, good, vgood

Requirement

We are required to calculate the 10-fold cross validation accuracy of two types of classifiers (A and B) on the two different datasets and find a value larger than 2 for the expression below

wine_acc(A) + car_acc(B) – wine_acc(B) – car_acc(A)

where *wine_acc(A)* refers to the accuracy of Classifier A on the wine data set, and *car_acc(B)* refers to the accuracy of Classifier B on the car data set, and so on.

We run a different set of classifiers separately on each dataset and try to analyze the results

Lazy IBK – K nearest neighbor Classifier:

K nearest neighbor is an instant based machine learning method. It is also called as lazy learning as the function is approximated locally and all computation is deferred until classification. It uses a distance measure to determine which class an instance belongs to. The value of k determines the number of neighbors to use while calculating the distance. We can change the value of k to different values in Weka and run the classifier. By default, it is set to 1.

Classifier	**Wine Accuracy**	**Car Accuracy**
Lazy IBK (k = 1)	86.8254%	90.1681%
Lazy IBK (k = 5)	84.6561%	90.1681%
Lazy IBK (k = 10)	83.3333%	84.3697%
Lazy IBK (k = 20)	83.8624%	78.5714%
Lazy IBK (k = 40)	83.4392%	78.5714%

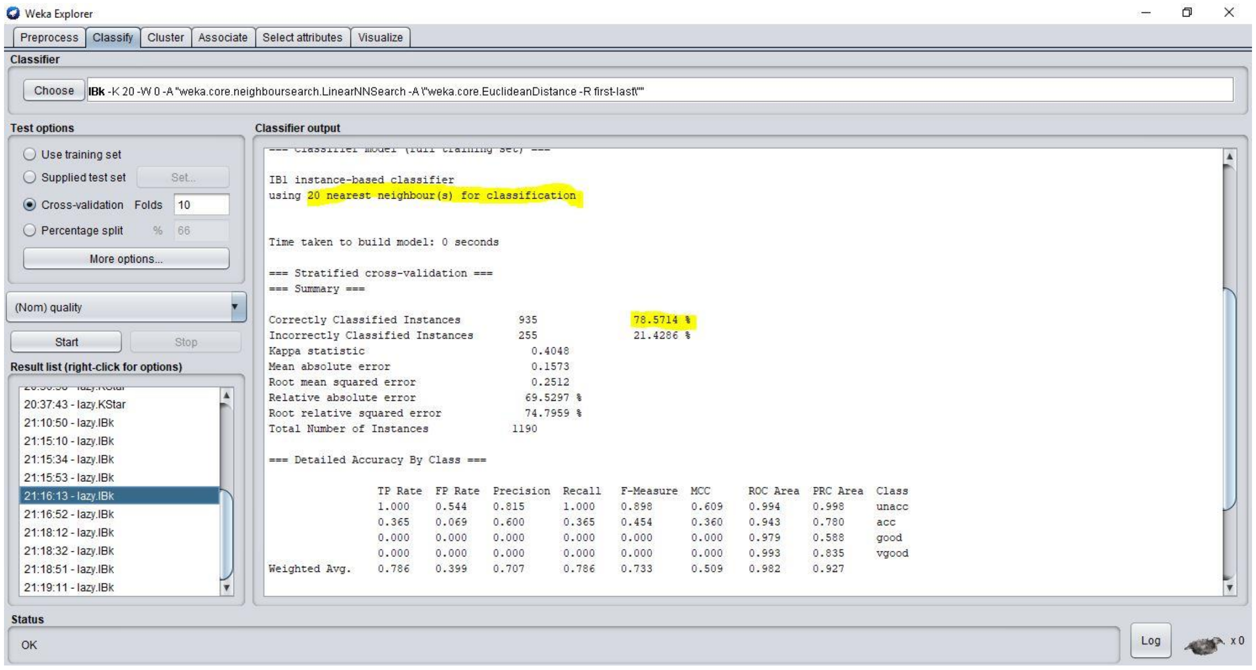
As the value of k increases, we can observe that the accuracy of the correctly classified instances in the car dataset gradually decrease. This implies that the car dataset provided is not noisy. In case of noisy data, the accuracy increases as the value of k becomes larger. However, it will always start to decrease if the value of k reaches an extreme value and settles down somewhere around the baseline accuracy for the model. We can also observe that the accuracy of wine does not change much and stabilizes around 83%.

We can choose this classifier with the value of k set to 20 as classifier A for solving our equation.

Classifier A – Lazy IBK, K-nearest neighbor algorithm, k = 20

win_acc(A) = 83.4392
car_acc(A) = 78.5714

Wine accuracy > Car accuracy



```
IBk -K 20 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
```

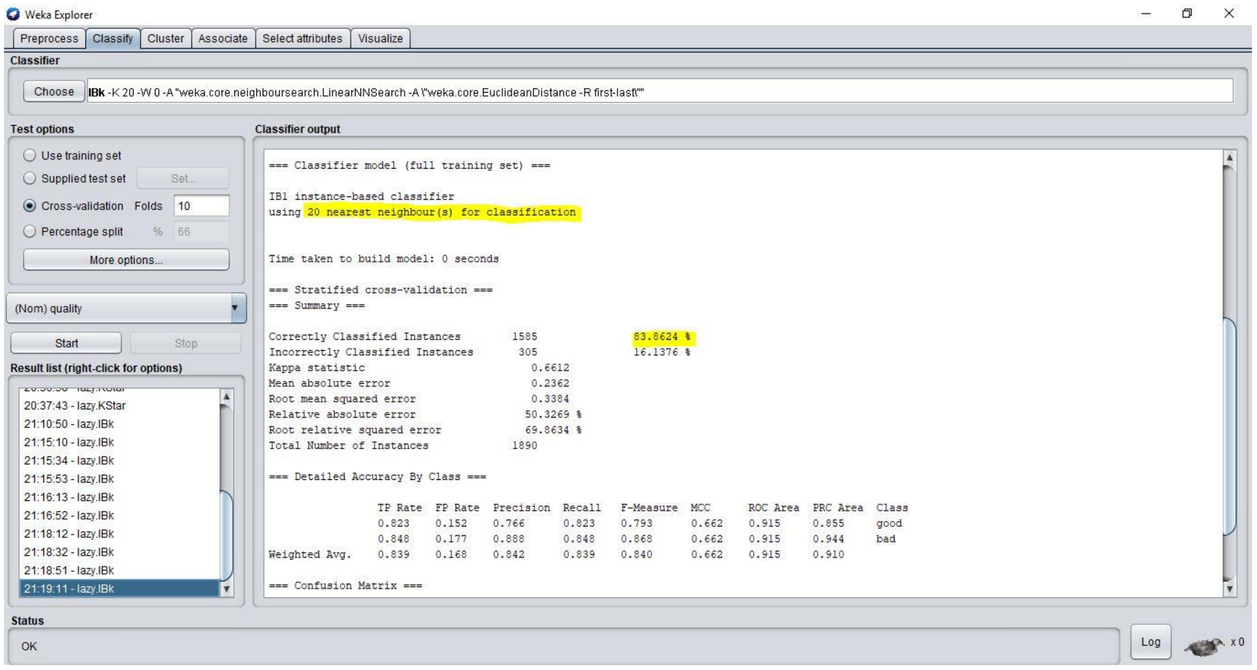
Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66
(Nom) quality

20:37:43 - lazy.KStar
21:10:50 - lazy.IBk
21:15:10 - lazy.IBk
21:15:34 - lazy.IBk
21:15:53 - lazy.IBk
21:16:13 - lazy.IBk
21:16:52 - lazy.IBk
21:18:12 - lazy.IBk
21:18:32 - lazy.IBk
21:18:51 - lazy.IBk
21:19:11 - lazy.IBk

```
IB1 instance-based classifier
using 20 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

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

Correctly Classified Instances         935               78.5714 %
Incorrectly Classified Instances       255               21.4286 %
Kappa statistic                          0.4048
Mean absolute error                      0.1573
Root mean squared error                  0.2512
Relative absolute error                 69.5297 %
Root relative squared error             74.7959 %
Total Number of Instances             1190

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.544    0.815      1.000    0.898      0.609    0.994     0.998     unacc
                 0.365    0.069    0.600      0.365    0.454      0.360    0.943     0.780     acc
                 0.000    0.000    0.000      0.000    0.000      0.000    0.979     0.588     good
                 0.000    0.000    0.000      0.000    0.000      0.000    0.993     0.835     vgood
Weighted Avg.    0.786    0.399    0.707      0.786    0.733      0.509    0.982     0.927
```

OK

Figure 23: Showing the wine accuracy for the k nearest neighbor classifier



Classifier: IBk -K 20 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

Test options: Cross-validation Folds 10

(Nom) quality

Result list (right-click for options):

```
20:37:43 - lazy.KStar
21:10:50 - lazy.IBk
21:15:10 - lazy.IBk
21:15:34 - lazy.IBk
21:15:53 - lazy.IBk
21:16:13 - lazy.IBk
21:16:52 - lazy.IBk
21:18:12 - lazy.IBk
21:18:32 - lazy.IBk
21:18:51 - lazy.IBk
21:19:11 - lazy.IBk
```

Classifier output:

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 20 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

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

Correctly Classified Instances        1585               83.8624 %
Incorrectly Classified Instances       305               16.1376 %
Kappa statistic                          0.6612
Mean absolute error                      0.2362
Root mean squared error                  0.3384
Relative absolute error                 50.3269 %
Root relative squared error             69.8634 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.823    0.152    0.766      0.823    0.793      0.662    0.915     0.855     good
                 0.848    0.177    0.888      0.848    0.868      0.662    0.915     0.944     bad
Weighted Avg.    0.839    0.168    0.842      0.839    0.840      0.662    0.915     0.910

=== Confusion Matrix ===
```

Status: OK

Figure 24: Showing the car accuracy for the k nearest neighbor classifier

Comparing the classifier results:

We now run a different set of classifiers on each dataset and calculate the 10-fold cross validation results. It is presented in the table below.

Classifier	**Wine Accuracy**	**Car Accuracy**
J48	85.9788%	89.8319%
Naïve Bayes	78.8889%	85.2941%
Random Tree	86.0317%	84.8257%
Multilayer Perceptron	84.9735%	99.2437%
Lazy KStar	87.4603%	87.395%

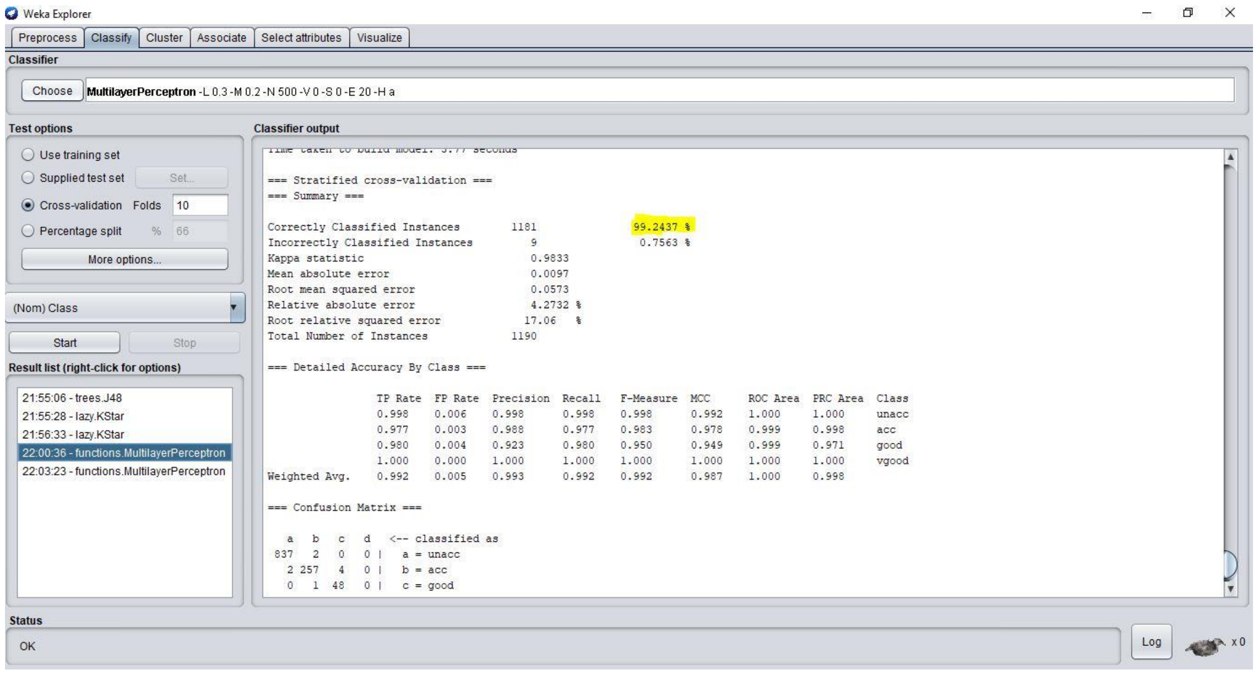
We can summarize the results as follows

- J48, Naïve Bayes and Multilayer Perceptron perform well on the car dataset and has a higher accuracy as compared to the wine dataset results
- Random Tree classifier works well on both the datasets with a higher accuracy for Wine
- Lazy K-star gives an approximately similar accuracy measures for both the datasets

We can also observe that the Multilayer Perceptron performs exceeding well with the car dataset. There is also a large difference between the accuracies of both the datasets. Even

Naïve-Bayes and J48 show a significant difference between the two dataset accuracies. We can choose any of the above three classifiers as Classifier B for our calculation purposes.

The Mutlilayer Perceptron is an artificial neural network that works exceedingly well on the car dataset. This performance can be attributed to the four class labels that the car evaluation dataset has as compared to the class labels for the wine dataset. However, the time taken to build these models is extremely high. To determine the largest value for the given expression we can use the MultilayerPerceptron as our second classifier.



Choose: MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

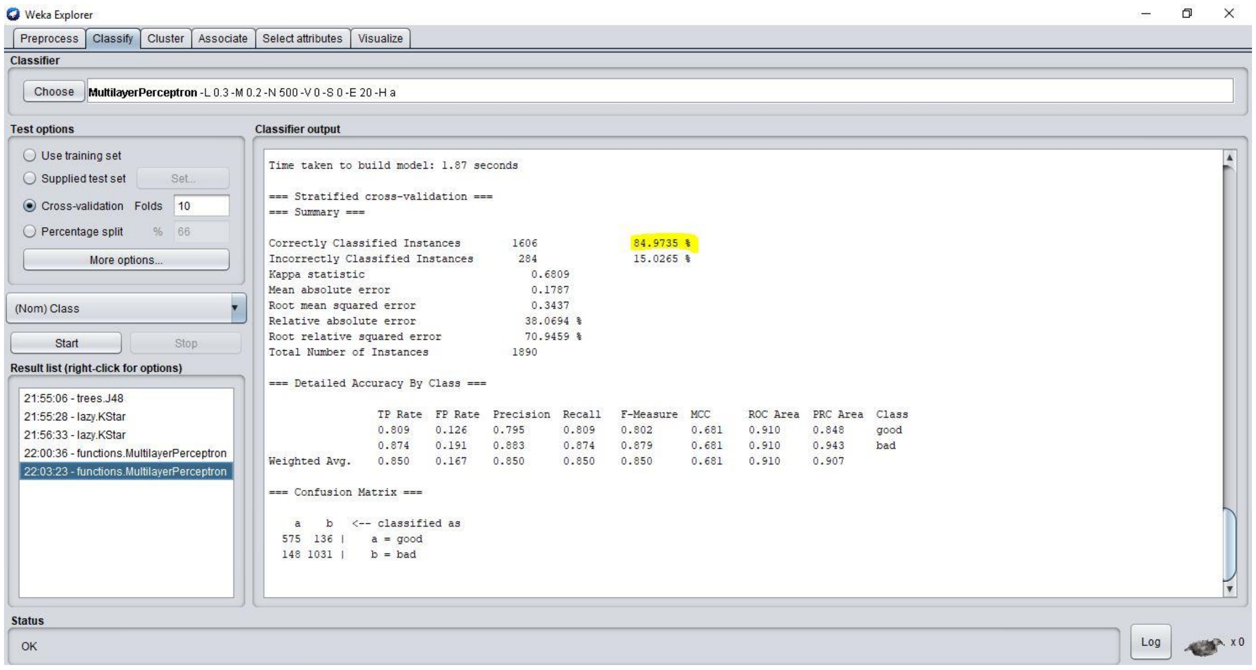
Test options: Cross-validation, Folds 10; Percentage split % 66

Result list:
21:55:06 - trees.J48
21:55:28 - lazy.KStar
21:56:33 - lazy.KStar
22:00:36 - functions.MultilayerPerceptron
22:03:23 - functions.MultilayerPerceptron

```
Time taken to build model: 3.77 seconds

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

Correctly Classified Instances        1181               99.2437 %
Incorrectly Classified Instances         9                0.7563 %
Kappa statistic                          0.9833
Mean absolute error                      0.0097
Root mean squared error                  0.0573
Relative absolute error                  4.2732 %
Root relative squared error             17.06   %
Total Number of Instances             1190

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.998    0.006    0.998      0.998    0.998      0.992    1.000     1.000     unacc
                 0.977    0.003    0.988      0.977    0.983      0.978    0.999     0.998     acc
                 0.980    0.004    0.923      0.980    0.950      0.949    0.999     0.971     good
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     vgood
Weighted Avg.    0.992    0.005    0.993      0.992    0.992      0.987    1.000     0.998

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 837   2   0   0 |   a = unacc
   2 257   4   0 |   b = acc
   0   1  48   0 |   c = good
```

Status: OK

Figure 25: showing the car accuracy for the MultilayerPerceptron classifier



Classifier: MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Test options: Cross-validation Folds 10; Percentage split % 66

Result list (right-click for options)

- 21:55:06 - trees.J48
- 21:55:28 - lazy.KStar
- 21:56:33 - lazy.KStar
- 22:00:36 - functions.MultilayerPerceptron
- 22:03:23 - functions.MultilayerPerceptron

Classifier output

```
Time taken to build model: 1.87 seconds

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

Correctly Classified Instances        1606               84.9735 %
Incorrectly Classified Instances       284               15.0265 %
Kappa statistic                          0.6809
Mean absolute error                      0.1787
Root mean squared error                  0.3437
Relative absolute error                 38.0694 %
Root relative squared error             70.9459 %
Total Number of Instances             1890

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.809    0.126    0.795      0.809    0.802      0.681    0.910     0.848     good
                 0.874    0.191    0.883      0.874    0.879      0.681    0.910     0.943     bad
Weighted Avg.    0.850    0.167    0.850      0.850    0.850      0.681    0.910     0.907

=== Confusion Matrix ===

    a    b   <-- classified as
  575  136 |    a = good
  148 1031 |    b = bad
```

Figure 26: Showing the wine accuracy for the MultilayerPerceptron classifier

Classifier B – Multilayer Perceptron
win_acc(B) = 84.9735%
car_acc(B) = 99.2437%

Wine accuracy < Car accuracy

We now calculate the result for the given expression

wine_acc(A) + car_acc(B) – wine_acc(B) – car_acc(A)

<u>83.4392 + 99.2437 – 84.9735 – 78.5714 = 19.138</u>

The value obtained for the given expression using the above classifiers is 19.138%.

Question #9:

Name one major difference between the output space for the car data set vs. the wine data set, that might make some classifiers that are applicable to the wine data not applicable to the car data.

We can identify the key difference between the output spaces by looking at the class attributes of both the datasets. We can clearly see that that there are four class values for the car evaluation dataset. This include the acceptability criteria for the car models. An instance can be counted as any of these class labels which include unacc, acc, good and vgood.

On the other hand, the wine dataset has only two class attributes, good and bad. This makes the classification of instances much simpler.

Furthermore, it can also be noted that the all the attributes provided in the wine dataset are numeric in nature, except the class attribute. Whereas, the attributes provided in the car dataset are nominal in nature.

J48 decision tree analysis:

When we conducted decision tree analysis on the wine dataset, the resulting tree structure was easy to read based on the numerical values for the separate attributes. For example, we can look at the value of alcohol at the root node and predict its class label based on that. If the value is less than 10.8, the quality of wine is more bad than good and vice-versa. The numerical values were also of great help while reading and understanding the histograms. We can predict the change in attribute affiliation based on the progression of the histogram in relation to the attributes numerical value on the x-axis.

However, since there are four different class values for the car dataset, the visualization will be difficult to comprehend. These class values are all nominal in nature.

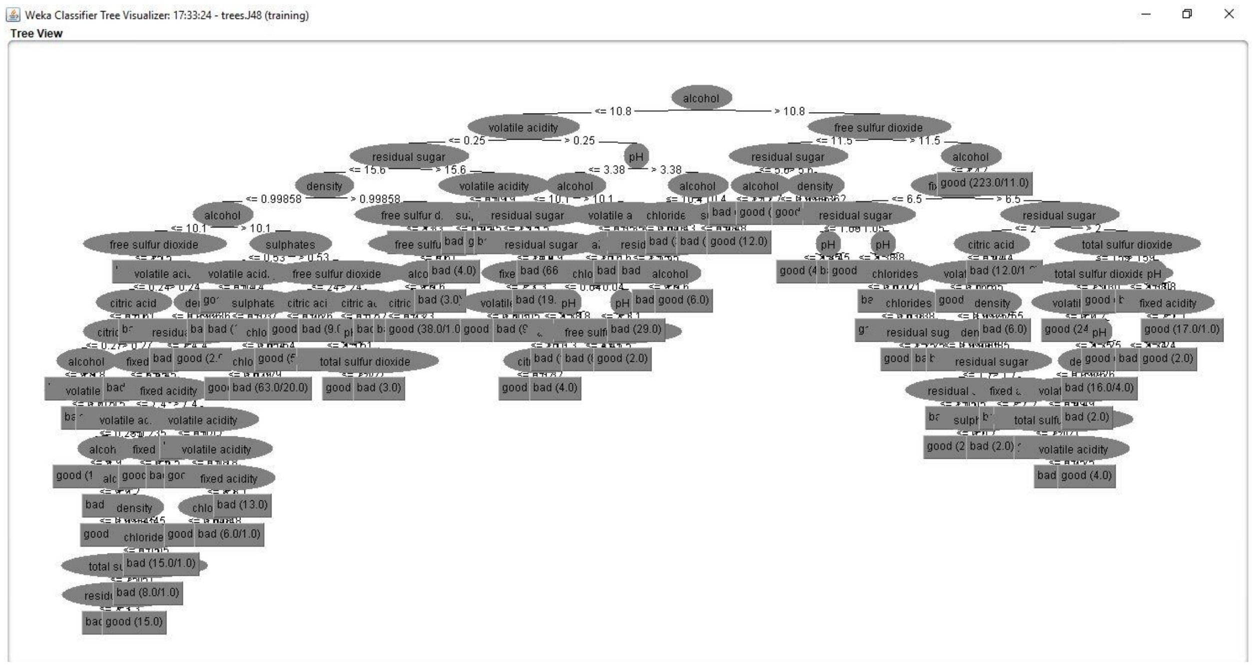


Figure 27: Tree Visualization for the wine dataset

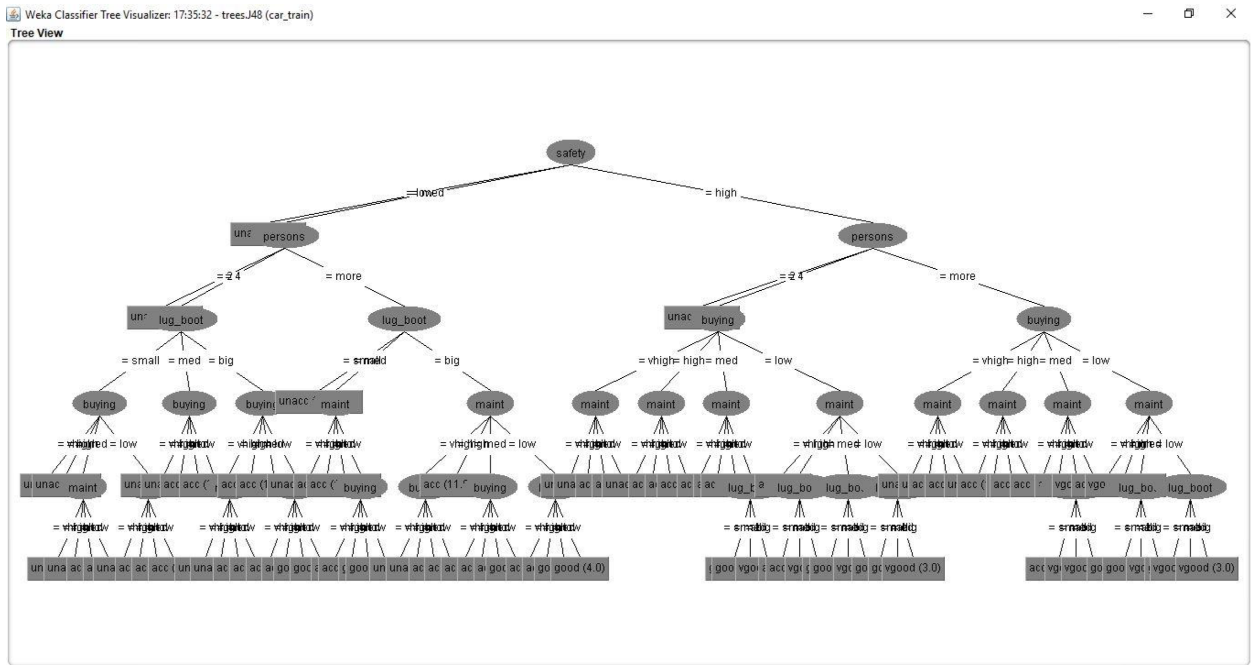


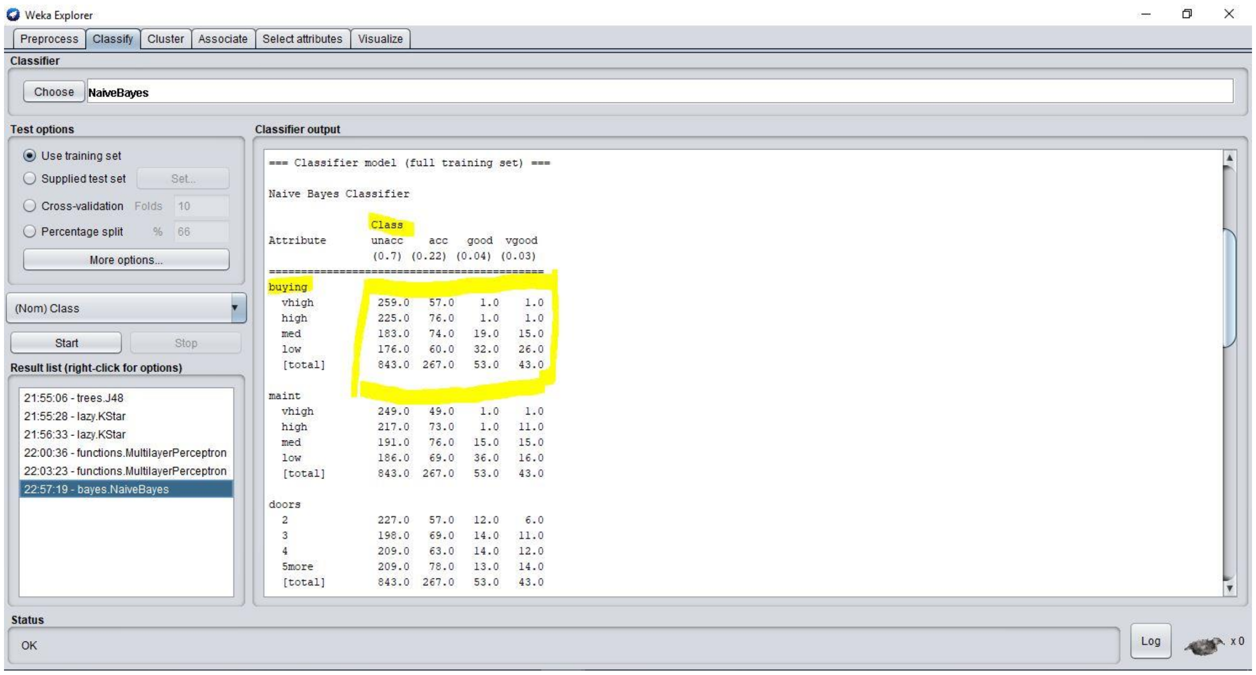
Figure 28: Tree visualization for the car dataset

As it can be observed, the tree visualization for the car dataset is much more difficult to comprehend because of the four class labels. In comparison, the tree visualization for wine dataset is much simpler to understand based on its numerical attributes and two class values, good and bad.

Naïve Bayes Classifier:

Another classifier that was efficient in predicting the car evaluation data as compared to the wine dataset was the Naïve-Bayes Algorithm.

When we analyze the results of this classifier on the training dataset for car, we can clearly count the number of instances of each individual attribute in relation to their class attributes. This makes it a definitive model for the car dataset. For example, we can look at the result and say how many 'vhigh' instances are 'unacc', 'acc', 'good' and 'vgood'. This can be observed in the figure below.



Classifier: NaiveBayes

Test options: (●) Use training set; ( ) Supplied test set; ( ) Cross-validation Folds 10; ( ) Percentage split % 66

(Nom) Class

Result list (right-click for options)

```
21:55:06 - trees.J48
21:55:28 - lazy.KStar
21:56:33 - lazy.KStar
22:00:36 - functions.MultilayerPerceptron
22:03:23 - functions.MultilayerPerceptron
22:57:19 - bayes.NaiveBayes
```

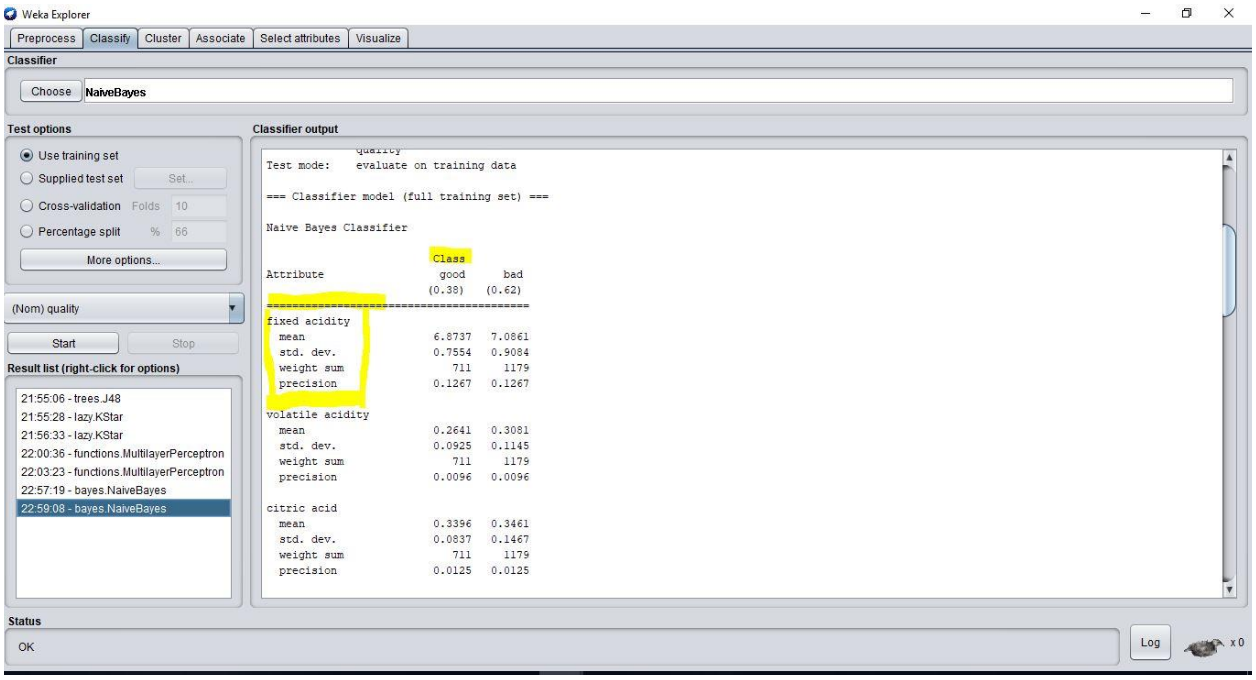
Classifier output

```
=== Classifier model (full training set) ===

Naive Bayes Classifier

                 Class
Attribute        unacc    acc   good  vgood
                 (0.7) (0.22) (0.04) (0.03)
===========================================
buying
  vhigh          259.0   57.0    1.0    1.0
  high           225.0   76.0    1.0    1.0
  med            183.0   74.0   19.0   15.0
  low            176.0   60.0   32.0   26.0
  [total]        843.0  267.0   53.0   43.0

maint
  vhigh          249.0   49.0    1.0    1.0
  high           217.0   73.0    1.0   11.0
  med            191.0   76.0   15.0   15.0
  low            186.0   69.0   36.0   16.0
  [total]        843.0  267.0   53.0   43.0

doors
  2              227.0   57.0   12.0    6.0
  3              198.0   69.0   14.0   11.0
  4              209.0   63.0   14.0   12.0
  5more          209.0   78.0   13.0   14.0
  [total]        843.0  267.0   53.0   43.0
```

Status: OK

Figure 29: Showing the number of instances of each attribute in relation to the 4 class attributes

However, when we run the classifier on the wine dataset, we can only observe the mean, standard deviation and other values in relation to the quality of wine which is good or bad.



Classifier: NaiveBayes

Test options: (●) Use training set; ( ) Supplied test set; ( ) Cross-validation Folds 10; ( ) Percentage split % 66

(Nom) quality

Result list (right-click for options)

```
21:55:06 - trees.J48
21:55:28 - lazy.KStar
21:56:33 - lazy.KStar
22:00:36 - functions.MultilayerPerceptron
22:03:23 - functions.MultilayerPerceptron
22:57:19 - bayes.NaiveBayes
22:59:08 - bayes.NaiveBayes
```

Classifier output

```
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Naive Bayes Classifier

                         Class
Attribute                 good      bad
                        (0.38)   (0.62)
=======================================
fixed acidity
  mean                  6.8737   7.0861
  std. dev.             0.7554   0.9084
  weight sum               711     1179
  precision             0.1267   0.1267

volatile acidity
  mean                  0.2641   0.3081
  std. dev.             0.0925   0.1145
  weight sum               711     1179
  precision             0.0096   0.0096

citric acid
  mean                  0.3396   0.3461
  std. dev.             0.0837   0.1467
  weight sum               711     1179
  precision             0.0125   0.0125
```

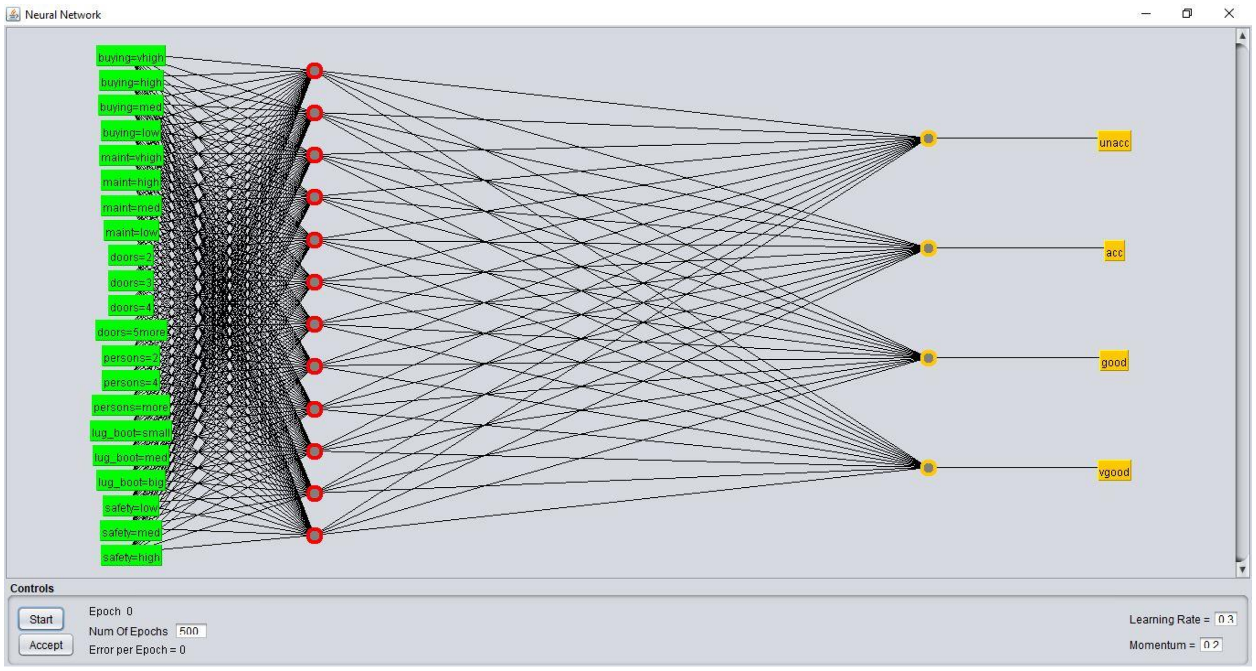
Status: OK

Figure 30: Showing the numerical values in relation to the class attributes

Naïve-Bayes classifier works extremely well in understanding the nature of data in the car dataset as compared to the wine dataset.

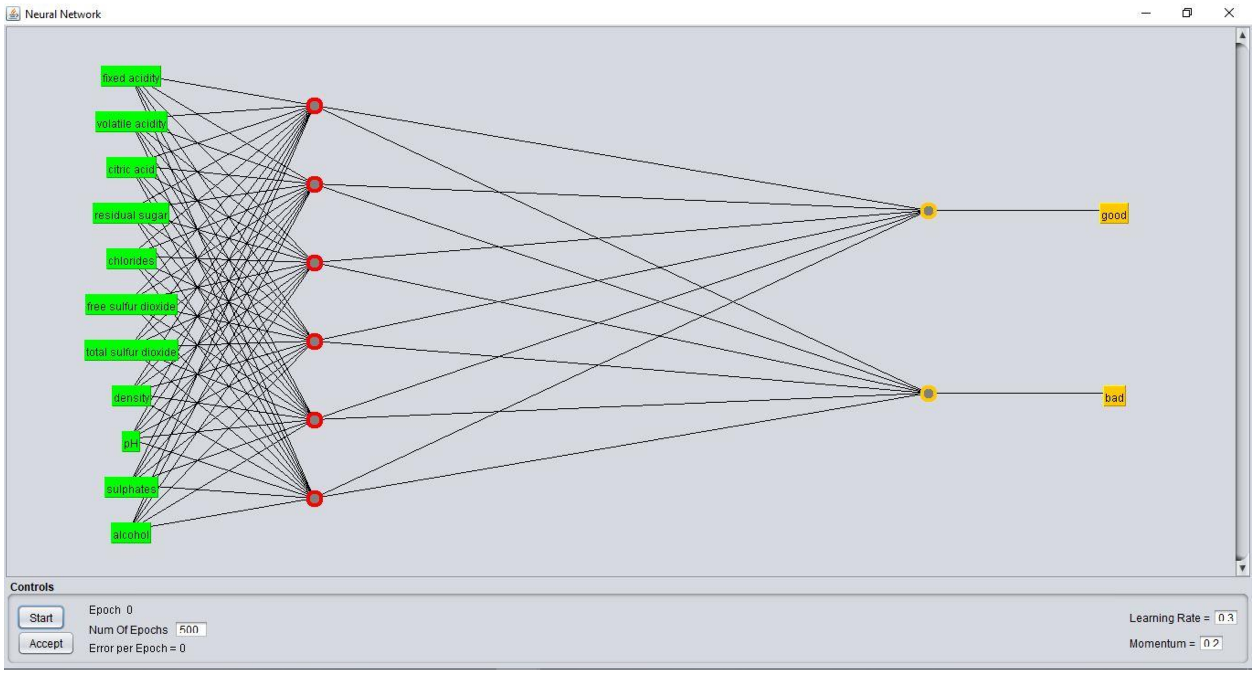
MultilayerPerceptron Classifier:

Another classifier that work extremely well on the car dataset as compared to the wine dataset is the MultilayerPerceptron. As we have four output classes in the car dataset, we can demonstrate the interconnection between the separate nodes using the artificial neural network of the multilayer Perceptron. This can be observed in the neural network GUI of the car dataset. The hidden nodes (displayed in red) connects the input nodes to the four output classes of the data.



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

Figure 31: Showing the neural network GUI for the car dataset



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

Figure 32: Showing the neural network for wine dataset with two class values

Summary:

By analyzing the above results, we can briefly summarize that the decision tree classifiers work extremely well with the wine data as compared to car. And the Naïve-Bayes can be used as a good model for the car dataset

Conclusion:

Data mining is an experimental science. There is no hard and fast rule to justify a certain rule or classifier for a certain dataset. Different classifiers tend to perform differently on multiple datasets. It all depends on the context and the information we are looking for in a chosen dataset.

Based on the aforesaid reasons, it would seem the key difference between the output spaces is that the car data has four different possible outputs, while the wine data has only two possible outputs. The number of output classes have a significant impact on the predictive nature of a classifier. As examined before, the same classifier may perform differently based on the number of output classes.