Post-Block Assignment 2

Francois van Zyl 18620426

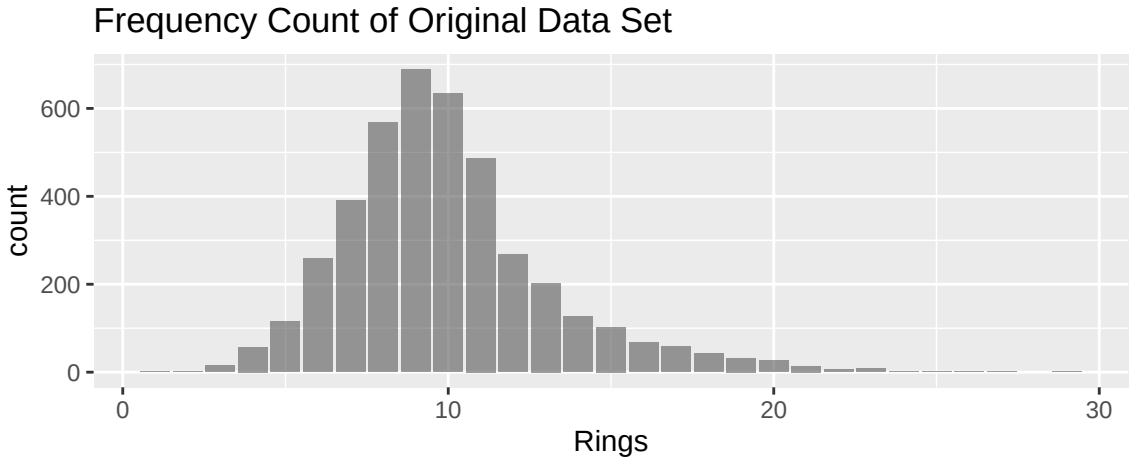
17 January 2021

Selection of Problem Type

Before deciding on whether to treat this as a multi-class classification or regression problem, I investigated a histogram and some descriptive statistics of the target feature Rings, as seen below.

Table 1: Descriptive Statistics and Histogram of Rings

Minimum	1st Quantile	Median	Mean	3rd Quantile	Maximum	Distinct
1	8	9	9.933685	11	29	28



All the entries for the target feature assume an integer value within the range $Rings \in \{1, 29\}$. 50% of the entries lie between $Rings \in \{8, 11\}$ and this imbalanced nature of the dataset will prove troublesome during the pruning process, as these lesser leaf nodes will undoubtedly be pruned away. However there are data resampling methods that can be used to balance the data set. I was interested in investigating this problem from a classification perspective with the usage of data resampling, and proceeded to use a classification tree.

Decision Tree Induction

Selection of Decision Tree

To select a decision tree, I first considered the nature of the current problem. As we are trying to predict a categorical value, some form of classification decision tree would be needed. One of the most well-known decision trees are CART, or Classification And Regression Trees. During training, these trees are constructed with a single root node, which is the feature that was calculated as being the most informative to split on in the data set. After finding this root node, decision nodes are formed by checking which features are the most informative to split on. This process of splitting is continued until leaf nodes are constructed with the target value in. Due to this, CART trees perform indirect feature selection as less informative features are simply left out.

To find the features that are most informative, these classification trees use measures to characterize the training observations' entropy. The feature that provides the highest reduction in entropy after splitting is selected as the most informative feature to split on. This entropy is used to characterize the disorder of the data set, and by minimizing it we are constructing a decision tree that ensures that more probable outcomes are grouped together. In the case of classification, these CART decision trees typically use either the Gini Impurity or Information Gain measures. These measures are expected to perform similarly, but for large data sets the Gini Impurity measure seems to be more computationally efficient and it is therefore preferable over the Information Gain measure.

Quoting the referenced software documentation, in the case of regression these CART decision trees will use the sum-of-squares of the node SS_T and the node's left and right children nodes, SS_L and SS_R, to decide on the feature that provides the best split at any given node. This method of splitting is the same as selecting a split that maximizes the inter-group sum-of-squares in an ANOVA test. Note that other methods exist as measures of splitting for regression, but this method seemed to be the most common choice in regression applications of CART.

Data Transformations

CART trees are sensitive to imbalanced classes. Like many decision trees, CART trees typically overfit a data set during training, and therefore a pruning process is applied after training to remove less important leaf nodes. In the presence of imbalanced classes, as well as outliers, this pruning proces might result in an underfitted tree, as the minority classes might be pruned away. This data set has multiple minority classes, and the number of instances falling into each unique category is displayed below. Please note that the upper row refers to the category, and the lower row refers to the count within the categories.

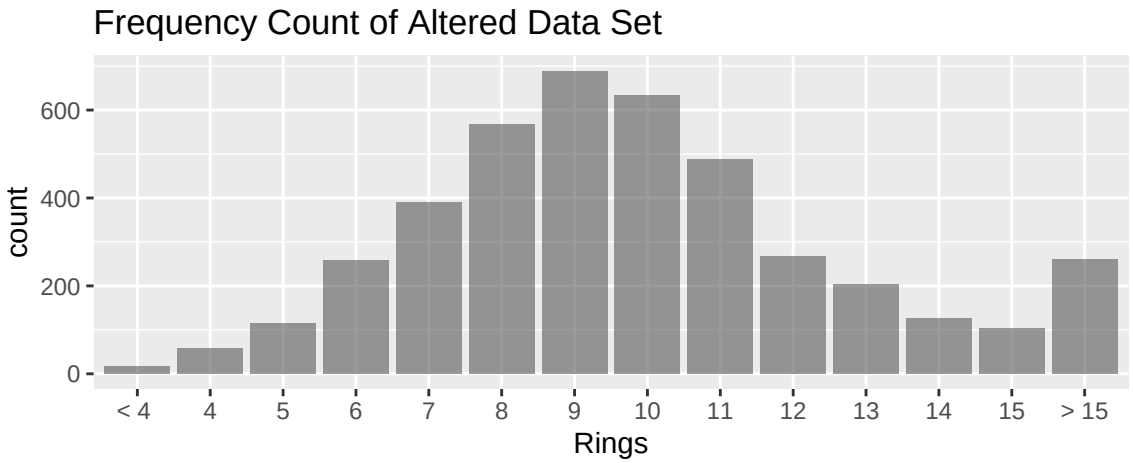
Table 2: Counts of Rings

1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	15	57	115	259	391	568	689	634	487	267	203

14	15	16	17	18	19	20	21	22	23	24	25	26	27	29
126	103	67	58	42	32	26	14	6	9	2	1	1	2	1

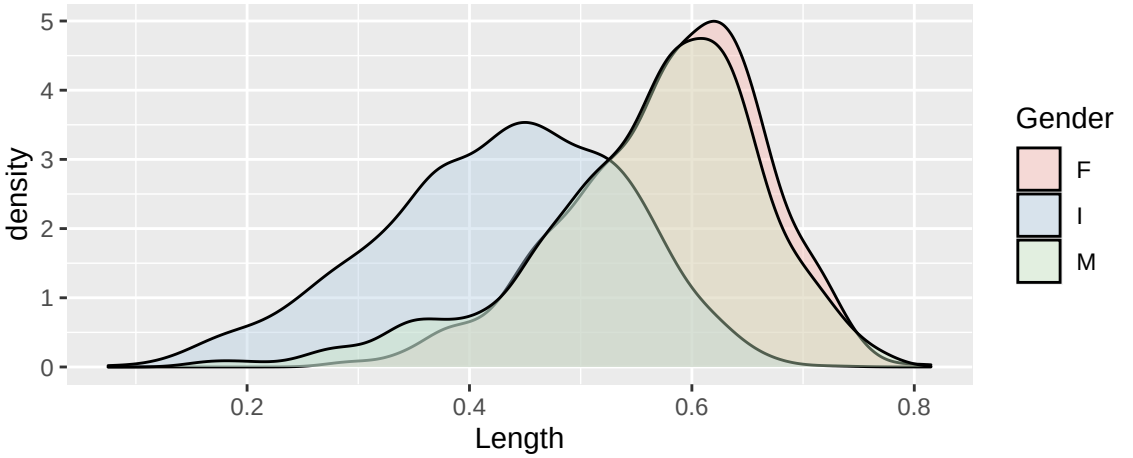
Note the scarcity of the data points between $Rings \in \{1,4\}$ and $Rings \in \{16,29\}$. In an attempt to combat this scarcity of data points, I decided to bin the lesser appearing categories into two new categories. These categories will represent all entries below and above a certain threshold of Rings. For example, instead of having $Rings = 21$, there will be a feature $Rings > 17$ that contains all features above the cut-off point.

This introduced a new problem, in that I had to decide what the cut-off point is for binning these values into these two new categories. For example, should anything with less than 100 entries be binned together, or should it be 50 entries, or 20 entries? In an attempt to find an answer for this, I decided to apply the IQR method on the Rings feature to identify outlying points, and then use this as indication of what range of the target feature to bucket together. By using anything outside 1.5 times the IQR range as an outlier, the minimum and maximum of the calculated borders are 4 and 15. Although this might be questionable, I binned all values *exceeding* these thresholds into two new categories. The newly created categories are displayed below in an histogram and a table that documents their respective entry counts.



< 4	4	5	6	7	8	9	10	11	12	13	14	15	> 15
17	57	115	259	391	568	689	634	487	267	203	126	103	261

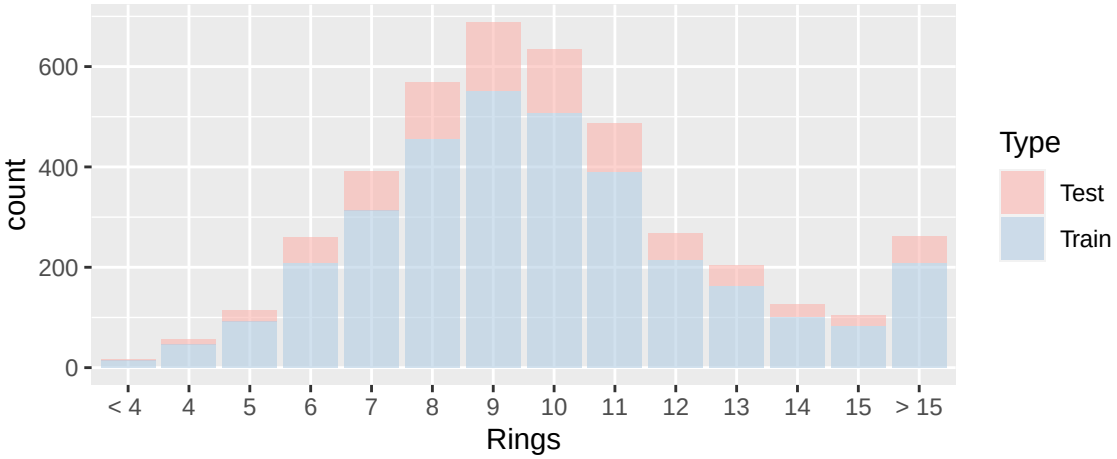
After introducing these two new categories, I merged the female and male categories from the Gender feature into one category called adult. I did this due to prior knowledge from the previous assignments, where we could recall that the distributions of the features are very similar for both male and female categories. This problem is better suited as an adult versus infant problem, and the distinction between the male and female categories could unnecessarily complicate the extracted rules (if they use the feature Gender). I did not plot the density plots across all features, but the density plot is shown below for the feature length, and we can see that the distributions are very similar for male and female.



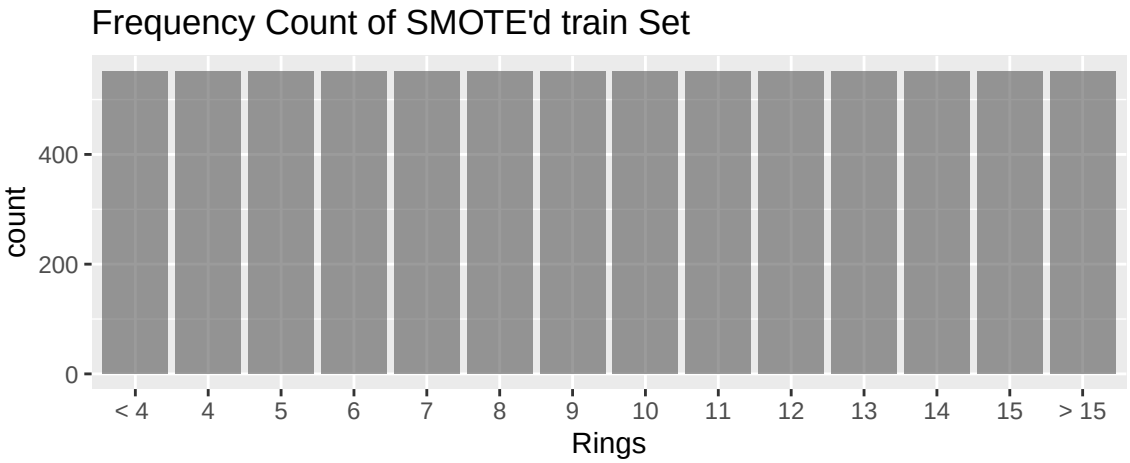
Before attempting any balancing of the data set, I first split the data set into training and testing sets as it is generally considered good practice to validate a model on authentic data. I used an 80/20 split and the resulting size of the training and testing sets are displayed below. The training and testing target value distributions are also displayed in a stacked bar plot below. The testing set seems to have a decent spread of the minority classes, and I proceeded to over-sample the training set. f

Table 5: Training and Testing Set: Before Oversampling

Train	Test
3348	829



To balance the data set, I performed the SMOTE technique on the training data set. As the data set contained nominal features I implemented the heterogeneous value difference metric as the distance measure, with 5 neighbors being used to generate the synthetic minority instances. I also applied a weighting on the oversampling such that the number of instances for all minority classes are equal to the number of instances for the target class 9. The SMOTE'd training data is displayed below. After this oversampling, the training set consisted of 7725 entries, with the original 9 features left intact.



CART Tree Induction

Before implementing the decision tree, I applied ten-fold cross validation to tune the decision tree's parameters. The parameters considered were the complexity parameter, the minimum number of observations that may exist in a leaf node, the minimum number of observations that must exist in a node for a split to be computed, the maximum depth of the decision tree, as well as the splitting criterion of the node.

Of all the tuned parameters, I found that the complexity parameter had the greatest influence on the tree performance. Quoting the software documentation, this complexity parameter relates the goodness-of-fit to whether a node should be split upon. It attempts to minimize the lack of goodness-of-fit with splits, and therefore it aids in deciding whether splits are truly useful, and it is a common measure used to combat overfitting. As the pruning process that I will be implementing uses this complexity parameter, I kept this complexity parameter constant at zero so that it does not affect the induction the process.

The results from the cross validation are displayed below. Note that only the top six parameter combinations are displayed. The accuracy metric was used to evaluate the performance of the tests, and the mean accuracy across the ten-fold cross validation tests are displayed on the right-most column. The complexity parameter can be seen to be left at zero, and the three numeric parameters were varied between 5, 10, and 20. The cross-validation test using the Gini method is displayed first, and it can be seen to perform slightly worse than the test using the information gain criterion. In fact, the three parameters did not change the accuracy of the validation tests much. The accuracy can be seen to be quite low, but I proceeded and selected the best parameters as displayed at the bottom of the page.

Table 6: Cross Validation With Gini Method

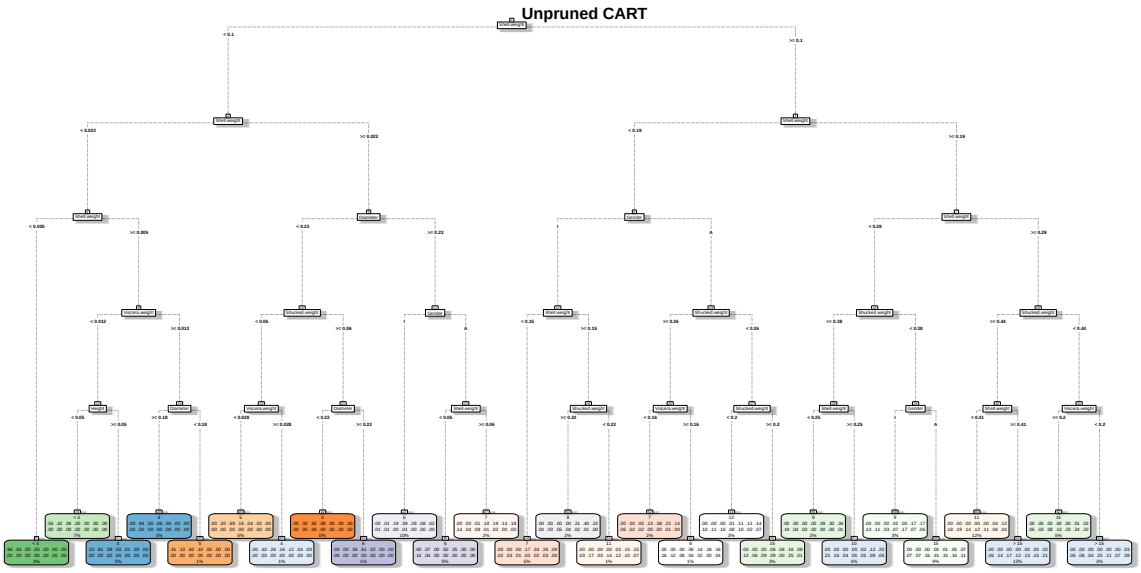
cp	minbucket	minsplit	maxdepth	acc.test.mean
0	10	5	5	0.3162509
0	10	10	5	0.3162509
0	10	20	5	0.3162509
0	20	5	5	0.3166393
0	20	10	5	0.3166393
0	20	20	5	0.3166393

Table 7: Cross Validation With Information Gain

cp	minbucket	minsplit	maxdepth	acc.test.mean
0	10	5	5	0.3207765
0	10	10	5	0.3207765
0	10	20	5	0.3207765
0	20	5	5	0.3214238
0	20	10	5	0.3214238
0	20	20	5	0.3214238

Table 8: Final Parameters Used

cp	minbucket	minsplit	maxdepth	split_criterion
0	10	5	5	Information



The induced decision tree is displayed above. My apologies for leaving the visualization and its' labels so small, the software implementation gave me quite a bit of trouble with the formatting. The graph is pixelated so if the examiner wishes to inspect it closer they should be able to see more if they zoom in.

We can see that there are a total of 27 leaf nodes, and therefore 27 rules, that can be extracted from this decision tree. Looking at the structure of the graph, there is one root node, 4 layers of decision nodes, and one layer of leaf nodes. To reach these leaf nodes, most paths seem to pass through all 4 layers of decision nodes. There are still some leaf nodes that only pass through 3 layers of decision nodes. By investigating the leaf nodes, we can see that the decision tree contains no leaf nodes that are capable of predicting class 13. The rest of the fourteen classes are all present in at least one leaf node.

The induced decision tree is displayed in text format below. In this text, each indentation represents a layer of nodes, and after passing through three or four layers of *decision nodes*, the text ends the rule with a colon, in which case the predicted value is given. For example, if the text displayed is: *[11] Diameter < 0.17939: 5 (n = 110, err = 37.3%)*, it first states the node number, *[11]*, then states a condition that has to be met at this node, *Diameter < 0.17939* and afterwards the predicted class label is given as *5*.

Then, the amount of observations present within this leaf node are displayed, which in this case is given as *n = 110*. This is equivalent to the support measure that will be considered later. After this, the amount of erroneous classifications are displayed as a percentage, that is the amount of observations that were placed in this leaf node whose true class label is not the same as the predicted class label. In this case, this is given as *err = 37.3%*. This represents the opposite of the accuracy metric, which is given by *1 - err*. This measure will be considered later, when the rule-based classifier is being discussed.

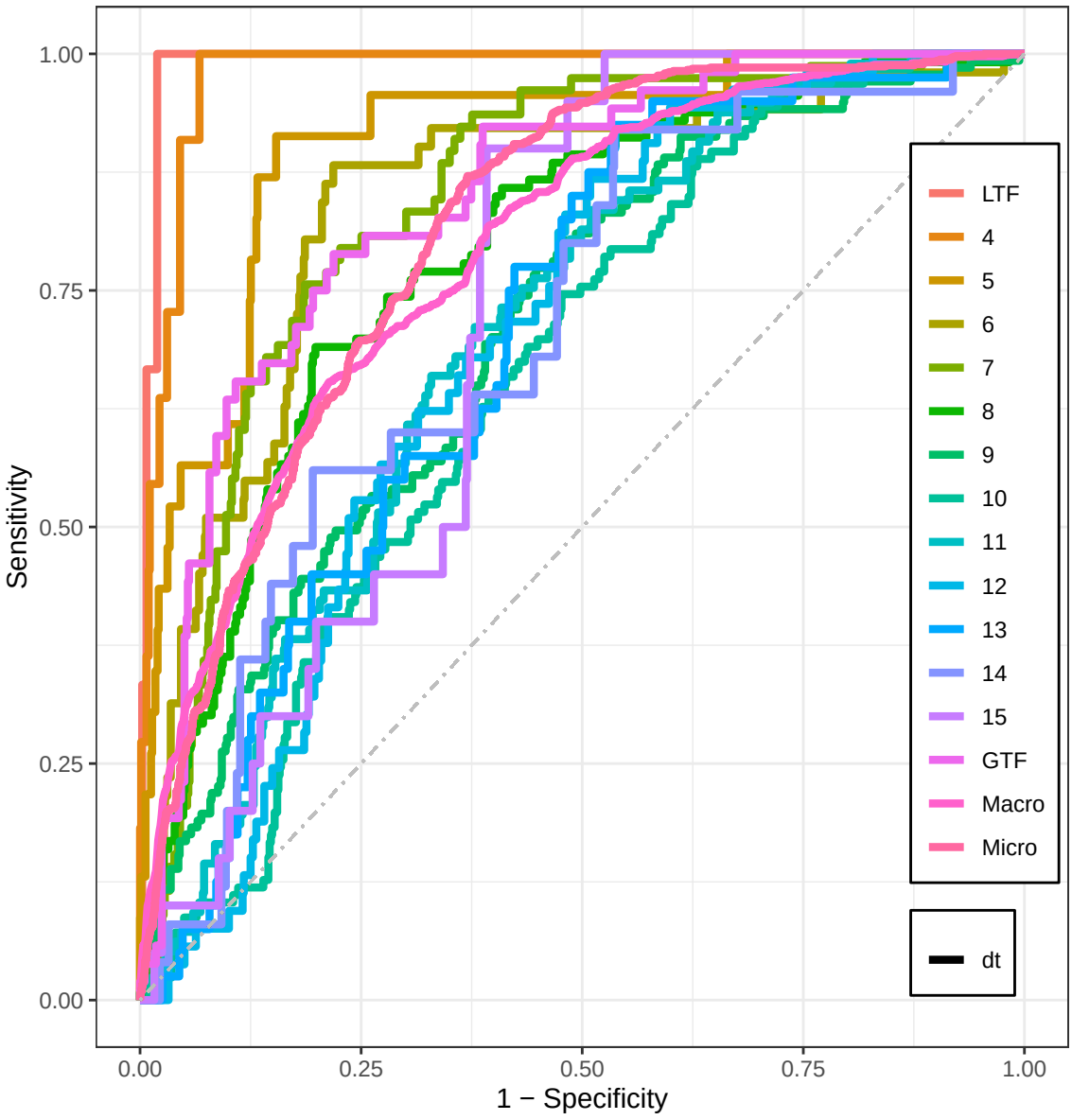
To further illustrate this misclassification error on the training data set, I will refer to the visualization of the unpruned decision tree above. In the leaf nodes, the actual class probabilities within these predicted class labels are displayed. For example, when inspecting the bottom right leaf node, that is the leaf node that predicts > 15 as the number of rings, we can see that the bottom right, or 14th, class percentage is given as 0.39. This value reflects the percentage of instances within this node that actually belong to the 14th class label, or > 15. Therefore, all other instances are incorrectly placed into this node and these class percentages illustrate what labels they had.

```
##
## Model formula:
## Rings ~ Gender + Length + Diameter + Height + Whole.weight +
##     Shucked.weight + Viscera.weight + Shell.weight
##
## Fitted party:
## [1] root
## |   [2] Shell.weight < 0.10386
## |   |   [3] Shell.weight < 0.0225
## |   |   |   [4] Shell.weight < 0.00496: < 4  (n = 232, err = 3.9%)
## |   |   |   [5] Shell.weight >= 0.00496
## |   |   |   |   [6] Viscera.weight < 0.01198
## |   |   |   |   |   [7] Height < 0.05027: < 4  (n = 571, err = 47.5%)
## |   |   |   |   |   [8] Height >= 0.05027: 4 (n = 124, err = 19.4%)
## |   |   |   |   [9] Viscera.weight >= 0.01198
## |   |   |   |   |   [10] Diameter >= 0.17939: 4 (n = 68, err = 5.9%)
## |   |   |   |   |   [11] Diameter < 0.17939: 5 (n = 110, err = 37.3%)
## |   |   [12] Shell.weight >= 0.0225
## |   |   |   [13] Diameter < 0.23488
## |   |   |   |   [14] Shucked.weight < 0.06008
## |   |   |   |   |   [15] Viscera.weight < 0.02801: 5 (n = 371, err = 41.0%)
## |   |   |   |   |   [16] Viscera.weight >= 0.02801: 4 (n = 111, err = 60.4%)
## |   |   |   |   [17] Shucked.weight >= 0.06008
## |   |   |   |   |   [18] Diameter < 0.21986: 5 (n = 36, err = 8.3%)
## |   |   |   |   |   [19] Diameter >= 0.21986: 6 (n = 44, err = 38.6%)
## |   |   |   [20] Diameter >= 0.23488
## |   |   |   |   [21] Gender in I: 6 (n = 735, err = 60.5%)
## |   |   |   |   [22] Gender in A
## |   |   |   |   |   [23] Shell.weight < 0.05989: 6 (n = 28, err = 50.0%)
## |   |   |   |   |   [24] Shell.weight >= 0.05989: 7 (n = 140, err = 80.7%)
## |   [25] Shell.weight >= 0.10386
## |   |   [26] Shell.weight < 0.19033
## |   |   |   [27] Gender in I
## |   |   |   |   [28] Shell.weight < 0.15027: 7 (n = 363, err = 65.6%)
## |   |   |   |   [29] Shell.weight >= 0.15027
## |   |   |   |   |   [30] Shucked.weight >= 0.2182: 8 (n = 133, err = 60.2%)
## |   |   |   |   |   [31] Shucked.weight < 0.2182: 11 (n = 59, err = 83.1%)
## |   |   |   [32] Gender in A
## |   |   |   |   [33] Shucked.weight >= 0.26275
## |   |   |   |   |   [34] Viscera.weight < 0.16102: 7 (n = 126, err = 61.9%)
## |   |   |   |   |   [35] Viscera.weight >= 0.16102: 8 (n = 49, err = 81.6%)
## |   |   |   |   [36] Shucked.weight < 0.26275
## |   |   |   |   |   [37] Shucked.weight < 0.20333: 12 (n = 246, err = 84.1%)
## |   |   |   |   |   [38] Shucked.weight >= 0.20333: 15 (n = 208, err = 74.5%)
## |   |   [39] Shell.weight >= 0.19033
## |   |   |   [40] Shell.weight < 0.29002
## |   |   |   |   [41] Shucked.weight >= 0.37994
## |   |   |   |   |   [42] Shell.weight < 0.25439: 9 (n = 189, err = 65.6%)
## |   |   |   |   |   [43] Shell.weight >= 0.25439: 10 (n = 306, err = 78.8%)
## |   |   |   |   [44] Shucked.weight < 0.37994
## |   |   |   |   |   [45] Gender in I: 9 (n = 215, err = 82.8%)
## |   |   |   |   |   [46] Gender in A: 15 (n = 732, err = 83.9%)
## |   |   |   [47] Shell.weight >= 0.29002
## |   |   |   |   [48] Shucked.weight >= 0.44499
```

```
## |   |   |   |   |   [49] Shell.weight < 0.40975: 11 (n = 942, err = 81.2%)
## |   |   |   |   |   [50] Shell.weight >= 0.40975: > 15 (n = 940, err = 78.7%)
## |   |   |   |   [51] Shucked.weight < 0.44499
## |   |   |   |   |   [52] Viscera.weight >= 0.20329: 15 (n = 417, err = 76.3%)
## |   |   |   |   |   [53] Viscera.weight < 0.20329: > 15 (n = 230, err = 61.3%)
##
## Number of inner nodes:    26
## Number of terminal nodes: 27
```

By inspecting the above text, we can see that the decision tree really struggled to model the data accurately. As expected, the classes that had to be up-sampled the most seem to have the lowest error in the training set, as seen in nodes [4], [8], [10] and [18].

To investigate the performance on the testing set, I first considered the AUC curve. The curve is displayed below, with a legend indicating the categories and the macro- and micro-averages obtained in the curve. From this plot can see that classes < 4 (LTF in legend), 4, and 5 achieved the best curves.



On the following page, two tables are displayed that order the categories by their obtained AUC value.

Table 9: Ordered AUC values: Testing Set

< 4	4	5	> 15	7	6	8	15	9	14	13	11	12	10
0.99	0.978	0.907	0.847	0.845	0.841	0.788	0.723	0.706	0.706	0.701	0.7	0.694	0.659

Table 10: Ordered AUC values: SMOTE'd Training Set

< 4	5	4	6	7	8	9	> 15	11	10	15	12	14	13
0.997	0.962	0.96	0.926	0.891	0.837	0.826	0.82	0.798	0.796	0.792	0.784	0.776	0.764

Using sources found online, $AUC > 0.85$ implies a good classification accuracy, $0.75 < AUC < 0.85$ implies a moderate classification accuracy, and $AUC < 0.75$ implies a low classification accuracy. We can therefore see that the decision tree could only predict categories *<4, 4, 5, >15, 7, 6, and 8* with moderate to good classification accuracy in the testing set. Some further testing set performance measures are displayed below.

Table 11: Performance Measures: Testing Set

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	F1	Balanced Accuracy
Class: < 4	0.2500000	0.9987820	0.6666667	0.9927361	0.3636364	0.6243910
Class: 4	0.2000000	0.9901720	0.2727273	0.9853301	0.2307692	0.5950860
Class: 5	0.3333333	0.9837297	0.4347826	0.9751861	0.3773585	0.6585315
Class: 6	0.3023256	0.9663526	0.5098039	0.9228792	0.3795620	0.6343391
Class: 7	0.3058824	0.9301075	0.3333333	0.9214381	0.3190184	0.6179949
Class: 8	0.4523810	0.8805591	0.1681416	0.9678771	0.2451613	0.6664700
Class: 9	0.3625000	0.8558077	0.2116788	0.9263006	0.2672811	0.6091539
Class: 10	0.2291667	0.8527529	0.0873016	0.9473684	0.1264368	0.5409598
Class: 11	0.2215569	0.9093656	0.3814433	0.8224044	0.2803030	0.5654612
Class: 12	0.0789474	0.9367889	0.0566038	0.9548969	0.0659341	0.5078681
Class: 13	NA	0.9517491	NA	NA	NA	NA
Class: 14	NA	0.9698432	NA	NA	NA	NA
Class: 15	0.0408163	0.9794721	0.3000000	0.8257108	0.0718563	0.5101442
Class: > 15	0.2771084	0.9611260	0.4423077	0.9227799	0.3407407	0.6191172

The sensitivity, or recall, relates to the decision tree's capacity to correctly classify a certain class. Classes with higher sensitivities are classified correctly more frequently, and we can see that the class 8 was classified correctly the best. Note that [7,11] were the dominant classes in the authentic data set. Of these dominant classes, classes 10 and 11 seem to be perform the worst in sensitivity. We can see that no classes were correctly classified as belonging to class 13 or 14.

The specificity is connected to the sensitivity, in that it relates to the decision tree's capacity to correctly classify a certain class as not belonging to this specified class. Classes with higher specificities are more often correctly classified as not belonging to a class than classes with lower specificities. We can see that the previously mentioned dominant classes all have the lowest specificities, and are therefore more commonly misclassified than other classes. This could be due to the authenticity of the data, as well as the nature of the data set. We can also see that classes 13 and 14 have an above-average specificity, despite never being correctly classified.

The positive predictive value measures the decision tree's probability that a class is assigned to the corresponding correct class, and the negative predictive value measures the probability that a class is correctly not assigned to a corresponding class. Classes 8, 10 and 12 have the lowest positive predictive value and

are therefore the least likely to be classified correctly. Similarly, by inspecting the negative predictive value, class 11 is more likely to be incorrectly assigned to other classes.

The F1 score takes into account both of these positive and negative predictive values, and according to this measure classes 12 and 15 perform the worst. This is corroborated by the balanced accuracy metric. The balanced accuracy metric provides a weighted accuracy, and takes into account the number of observations for each class. According to this measure classes 12 and 15 performed the worst.

After investigating these measures, I considered the accuracy metric of the whole classification problem on the testing set. This was calculated and is displayed below. This is a very low value, especially considering that the skew classes were balanced.

Table 12: Accuracy measure: Unpruned Decision Tree

acc
0.2352232

Rule Extraction

I proceeded to extract rules from the CART tree. The rules are displayed in a table below with the specified format of **if antecedent then consequent**. The antecedent for each rule is given by the respective statements under the column labeled IF. Similarly, the consequent for each rule is given by the value under the column labeled THEN. The support, or amount of instances that satisfy the antecedent of the rule, is displayed. The coverage is also displayed, and it is equivalent to the percentage of supporting instances within the training set that satisfies the antecedent. The accuracy of the rules are also displayed, and this measure reflects the fraction of instances that meet the antecedent and are correctly classified as having the associated consequent. Note both accuracy and coverage were left in their fraction forms, and they are not displayed in percentages.

Table 13: Rules Extracted

ID	IF	THEN	Support	Coverage	Accuracy
1	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight < 0.00496442712005228	< 4	232	0.03003	0.9573
2	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight < 0.0119823484576773 & Height < 0.0502689462387934	< 4	571	0.07392	0.5253
3	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight < 0.0119823484576773 & Height >= 0.0502689462387934	4	124	0.01605	0.8016
4	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight >= 0.0119823484576773 & Diameter >= 0.179393477778067	4	68	0.008803	0.9286

ID	IF	THEN	Support	Coverage	Accuracy
5	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight >= 0.0119823484576773 & Diameter < 0.179393477778067	5	110	0.01424	0.625
6	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter < 0.234877825158183 & Shucked.weight < 0.0600758844270604 & Viscera.weight < 0.028007330490509	5	371	0.04803	0.5898
7	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter < 0.234877825158183 & Shucked.weight < 0.0600758844270604 & Viscera.weight >= 0.028007330490509	4	111	0.01437	0.3982
8	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter < 0.234877825158183 & Shucked.weight >= 0.0600758844270604 & Diameter < 0.219856762643321	5	36	0.00466	0.8947
9	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter < 0.234877825158183 & Shucked.weight >= 0.0600758844270604 & Diameter >= 0.219856762643321	6	44	0.005696	0.6087
10	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter >= 0.234877825158183 & Gender %in% c('I')	6	735	0.09515	0.3948
11	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter >= 0.234877825158183 & Gender %in% c('A') & Shell.weight < 0.0598933696160093	6	28	0.003625	0.5
12	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter >= 0.234877825158183 & Gender %in% c('A') & Shell.weight >= 0.0598933696160093	7	140	0.01812	0.1972
13	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('I') & Shell.weight < 0.150274034830101	7	363	0.04699	0.3452

ID	IF	THEN	Support	Coverage	Accuracy
14	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('I') & Shell.weight >= 0.150274034830101 & Shucked.weight >= 0.218196077297325	8	133	0.01722	0.4
15	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('I') & Shell.weight >= 0.150274034830101 & Shucked.weight < 0.218196077297325	11	59	0.007638	0.1803
16	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight >= 0.26275 & Viscera.weight < 0.161016628616257	7	126	0.01631	0.3828
17	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight >= 0.26275 & Viscera.weight >= 0.161016628616257	8	49	0.006343	0.1961
18	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight < 0.26275 & Shucked.weight < 0.203326618075371	12	246	0.03184	0.1613
19	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight < 0.26275 & Shucked.weight >= 0.203326618075371	15	208	0.02693	0.2571
20	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight >= 0.37994221862976 & Shell.weight < 0.254386284024222	9	189	0.02447	0.3455
21	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight >= 0.37994221862976 & Shell.weight >= 0.254386284024222	10	306	0.03961	0.2143
22	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight < 0.37994221862976 & Gender %in% c('I')	9	215	0.02783	0.1751
23	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight < 0.37994221862976 & Gender %in% c('A')	15	732	0.09476	0.1621

ID	IF	THEN	Support	Coverage	Accuracy
24	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight >= 0.444987597530009 & Shell.weight < 0.40975	11	942	0.1219	0.1886
25	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight >= 0.444987597530009 & Shell.weight >= 0.40975	> 15	940	0.1217	0.2134
26	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight < 0.444987597530009 & Viscera.weight >= 0.203288824349467	15	417	0.05398	0.2387
27	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight < 0.444987597530009 & Viscera.weight < 0.203288824349467	> 15	230	0.02977	0.3879

By inspecting the extracted rules, and investigating the rules with the highest and lowest accuracies, we can conclude which rules are the best and worst at accurately classifying the training set. The following tables display the ten rules that obtained the highest and lowest accuracies in the training set.

Table 14: Rules: Highest accuracies

ID	1	4	8	3	5	9	6	2	11	14
THEN	< 4	4	5	4	5	6	5	< 4	6	8
Accuracy	0.957	0.929	0.895	0.802	0.625	0.609	0.590	0.525	0.500	0.400

Table 15: Rules: Lowest accuracies

ID	19	26	21	25	12	17	24	15	22	23	18
THEN	15	15	10	> 15	7	8	11	11	9	15	12
Accuracy	0.257	0.239	0.214	0.213	0.197	0.196	0.189	0.180	0.175	0.162	0.161

By re-inspecting Node [4] on page 7, that is the node corresponding to < 4, we can see that in terms of misclassification error, this leaf node performed the best in the entire decision tree. In the extracted rule set, the rule relating to this node (ID = 1) performed the best in terms of accuracy. The accuracies achieved across all these rules reflect the opposites of the misclassification errors illustrated in the decision tree.

We can see that the three rules with ID 1, 4 and 8; performed the best in accurately classifying the three classes < 4, 4, and 5. These were also the classes with the highest specificities, and within the training data set they seem to be the easiest to distinguish from other classes. The classes with the lowest accuracies correspond to classes 11, 9, 15, and 12. This corresponds with what was found with the decision tree. I then

proceeded to investigate the coverage of these rules.

Table 16: Rules: Highest coverage

ID	25	24	10	23	2	26	6	13	21	18
THEN	> 15	11	6	15	< 4	15	5	7	10	12
Coverage	0.122	0.122	0.095	0.095	0.074	0.054	0.048	0.047	0.040	0.032

Table 17: Rules: Lowest coverage

ID	14	3	16	5	7	4	15	9	17	8	11
THEN	8	4	7	5	4	4	11	6	8	5	6
Coverage	0.017	0.016	0.016	0.014	0.014	0.009	0.008	0.006	0.006	0.005	0.004

The rules with the highest coverage are displayed above. The maximum coverage is obtained by the class $>$ 15, with a value of 12.2%. The lowest coverage is 0.4%, and in fact there are 6 rules that do not even achieve a coverage of > 1 %. This is likely overfitting, and the pruning process will most likely remove these rules. This will be investigated later. Due to the nature of the rule extraction, this rule-based classifier performed exactly the same on the testing set as the decision tree and achieved an overall accuracy of 0.2352232.

Furthermore, the rules extracted from the decision tree are quite complicated and the majority of them are constructed from the AND operator of five IF statements. This is because when traversing the decision tree, most paths to leaf nodes pass through the root node and either three or four layers of decision nodes. The rules are quite complicated, and can be seen to heavily rely on the feature shell weight. This is especially visible when inspecting the extracted rule with ID $= 1$, as it splits on shell weight three times before reaching its leaf node.

Pruning Process

To perform the pruning process on the rule set, I performed an approach based on pruning the induced decision tree. First, I performed a ten-fold cross-validation test on the initial induced decision tree to find the complexity parameter at its current parameters [minsplit = 5, maxdepth = 5, minbucket = 10]. The results of this cross validation test are displayed below, and the complexity parameter can be seen to achieve a value that minimizes the relative error of the decision tree. The value of this complexity parameter is displayed below.

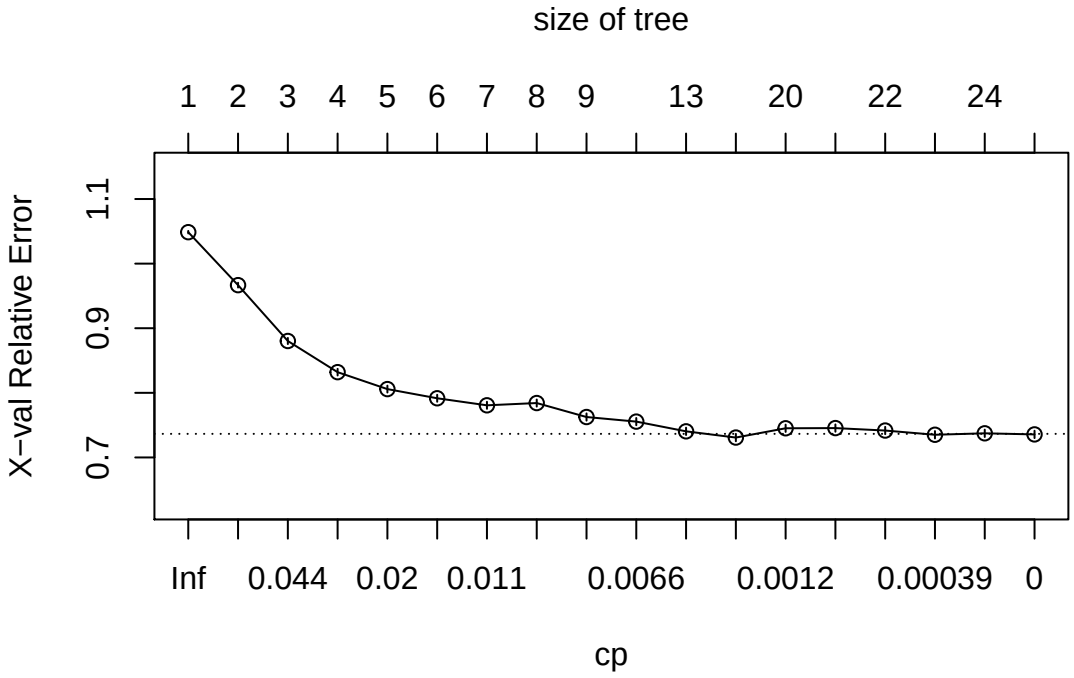


Table 18: Complexity Parameter

x
0.0012082

After finding this complexity parameter for the decision tree, I pruned the induced decision tree according to it. I then proceeded to extract new rules from this pruned decision tree, and used this new set of rules as my pruned rule set. The extracted rules are displayed below.

Table 19: Rules Extracted

ID	IF	THEN	Support	Coverage	Accuracy
1	Shell.weight $<$ 0.103855978908599 & Shell.weight $<$ 0.0224974514052155 & Shell.weight $<$ 0.00496442712005228	$<$ 4	232	0.03003	0.9573

ID	IF	THEN	Support	Coverage	Accuracy
2	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight < 0.0119823484576773 & Height < 0.0502689462387934	< 4	571	0.07392	0.5253
3	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight < 0.0119823484576773 & Height >= 0.0502689462387934	4	124	0.01605	0.8016
4	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight >= 0.0119823484576773 & Diameter >= 0.179393477778067	4	68	0.008803	0.9286
5	Shell.weight < 0.103855978908599 & Shell.weight < 0.0224974514052155 & Shell.weight >= 0.00496442712005228 & Viscera.weight >= 0.0119823484576773 & Diameter < 0.179393477778067	5	110	0.01424	0.625
6	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter < 0.234877825158183	5	562	0.07275	0.5284
7	Shell.weight < 0.103855978908599 & Shell.weight >= 0.0224974514052155 & Diameter >= 0.234877825158183	6	903	0.1169	0.3646
8	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('I') & Shell.weight < 0.150274034830101	7	363	0.04699	0.3452
9	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('I') & Shell.weight >= 0.150274034830101	8	192	0.02485	0.3247
10	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight >= 0.26275	7	175	0.02265	0.3164
11	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight < 0.26275 & Shucked.weight < 0.203326618075371	12	246	0.03184	0.1613
12	Shell.weight >= 0.103855978908599 & Shell.weight < 0.1903277044805 & Gender %in% c('A') & Shucked.weight < 0.26275 & Shucked.weight >= 0.203326618075371	15	208	0.02693	0.2571

ID	IF	THEN	Support	Coverage	Accuracy
13	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight >= 0.37994221862976	9	495	0.06408	0.2555
14	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight < 0.290022509438568 & Shucked.weight < 0.37994221862976	14	947	0.1226	0.156
15	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight >= 0.444987597530009 & Shell.weight < 0.40975	11	942	0.1219	0.1886
16	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight >= 0.444987597530009 & Shell.weight >= 0.40975	> 15	940	0.1217	0.2134
17	Shell.weight >= 0.103855978908599 & Shell.weight >= 0.1903277044805 & Shell.weight >= 0.290022509438568 & Shucked.weight < 0.444987597530009	> 15	647	0.08375	0.2804

The first thing to notice is that 10 rules were pruned away, as the rule set size went from 27 to 17 after the pruning process. I expect some of these rules that were pruned away to be the rules with the lowest coverage. We can check by inspecting the table below. After seeing the minimum coverage achieved, and by referring back to the unpruned rule set on page 15, we can see that the five rules corresponding to ID's 15, 9, 17, 8, and 11 were pruned out of the data set. These five rules had very low coverages, and were overfit on the training set.

Table 20: Pruned Rules: Lowest coverage

ID	6	13	8	11	1	12	9	10	3	5	4
THEN	5	9	7	12	< 4	15	8	7	4	5	4
Coverage	0.073	0.064	0.047	0.032	0.030	0.027	0.025	0.023	0.016	0.014	0.009

This can be investigated closer when noting that in the pruned rule set, there exist only two rules capable of predicting class label 5, namely ID's 5 and 6, with respective coverages of 0.01424 and 0.07275 and accuracies of 0.625 and 0.5898. In the unpruned rule set, there exist three rules capable of predicting class label 5, namely ID's 5, 6, and 8 with respective coverages of 0.01424, 0.04803 and 0.00466 and accuracies of 0.625, 0.5898 and 0.8947. Due to ID 8's low coverage and its' subpar contribution to generalization performance, it was removed during the pruning process.

We can also see that the majority of rules extracted are less complicated now. For instance, with the unpruned rule set note that on page 11 there is only one rule that utilizes only three if statements in its' antecedent, as seen in the rule with ID 1. All other rules utilize the AND operator of four or five IF statements. In the pruned rule set, there now exist three rules that only require three if statements in their antecedents; namely ID's 1, 6 and 7 with coverages 0.03003, 0.04803 and 0.00466 and accuracies 0.9573, 0.5898, and 0.8947. Similarly, in the pruned rule set, there are 8 rules that utilize 5 if statements. In the unpruned rule set, there were 24 rules that utilized 5 if statements. The unpruned rule therefore contains more complicated rules, and the pruning process reduced the complexity of the rule set. This can be illustrated further by investigating the amount of rules and their respective rule lengths, as seen in the table below.

Table 21: Complexity of Extracted Rules

Complexity	Unpruned	Pruned
3	1	3
4	2	6
5	24	8

The complexity column refers to the amount of statements used in the antecedent, and it can be seen that the pruned data set not only had more rules that used only 3 and 4 statements, but it also had significantly less rules that used 5 if statements. This undoubtedly reduced the performance of the extracted rules on the training set, but it should hopefully increase the generalization performance.

To investigate the generalization performance of the testing set, I first investigated the performance of the rules with regards to the accuracy and AUC metric. These measures are displayed below.

Table 22: Accuracy measure: Pruned Rule Set

acc
0.2484922

Table 23: Pruned Rules: AUC values: Testing Set

< 4	4	5	> 15	6	7	15	8	14	11	13	9	10	12
0.99	0.976	0.906	0.853	0.843	0.836	0.789	0.774	0.757	0.693	0.692	0.69	0.658	0.656

The accuracy is still quite low, however it is slightly better [1.3%] than the initial decision tree as seen on page 11. The AUC metric also revealed that the pruned rule set is much better at classifying class 14. There are also some rules that the AUC metric seems to have gotten worse at classifying, such as class 12. I then proceeded to investigate the same performance measures that were covered with the decision tree as seen below.

Initially, the unpruned decision tree did not predict any instances as belonging to class labels 13 and 14. The pruned rule set differed, in that it did not predict any instances belonging to classes 10 and 13. I will now briefly discuss these performance measures. The sensitivity changed somewhat between the decision tree and the pruned rule set. Classes 4, 8 and 9 seemed to gain the most from the pruning process, as they performed better than the decision tree. These classes were therefore classified correctly more often. Similarly, classes 5 and 6 seemed to perform worse after pruning.

Table 24: Sensitivity

	Pruned Rules	Decision Tree
Class: < 4	0.2500000	0.2500000
Class: 4	0.3333333	0.2000000
Class: 5	0.3076923	0.3333333
Class: 6	0.2500000	0.3023256
Class: 7	0.2916667	0.3058824
Class: 8	0.5000000	0.4523810
Class: 9	0.4021739	0.3625000
Class: 10	NA	0.2291667
Class: 11	0.2327044	0.2215569
Class: 12	0.0789474	0.0789474
Class: 13	NA	NA
Class: 14	0.0781250	NA
Class: 15	0.0476190	0.0408163
Class: > 15	0.2820513	0.2771084

The specificity is displayed below, and there were very marginal differences between the decision tree and the pruned rule set. Class 9 was the only class that showed a change of greater than 1%, and instances were therefore classified incorrectly as class 9 less often.

Table 25: Specificity

	Pruned Rules	Decision Tree
Class: < 4	0.9987820	0.9987820
Class: 4	0.9902439	0.9901720
Class: 5	0.9860759	0.9837297
Class: 6	0.9655172	0.9663526
Class: 7	0.9247028	0.9301075
Class: 8	0.8831004	0.8805591
Class: 9	0.8643148	0.8558077
Class: 10	0.8480097	0.8527529

	Pruned Rules	Decision Tree
Class: 11	0.9104478	0.9093656
Class: 12	0.9367889	0.9367889
Class: 13	0.9517491	0.9517491
Class: 14	0.9786020	0.9698432
Class: 15	0.9764851	0.9794721
Class: > 15	0.9733146	0.9611260

The positive predictive value is displayed below, and we can see a positive change in classes 5, 8, 9, 14 and a significant change in > 15 for the pruned rule set. However, class 15 has a very low positive predictive value and it performed worse than the initial decision tree.

Table 26: Pos Pred Value

	Pruned Rules	Decision Tree
Class: < 4	0.6666667	0.6666667
Class: 4	0.2727273	0.2727273
Class: 5	0.5217391	0.4347826
Class: 6	0.5098039	0.5098039
Class: 7	0.2692308	0.3333333
Class: 8	0.1858407	0.1681416
Class: 9	0.2700730	0.2116788
Class: 10	NA	0.0873016
Class: 11	0.3814433	0.3814433
Class: 12	0.0566038	0.0566038
Class: 13	NA	NA
Class: 14	0.4000000	NA
Class: 15	0.0500000	0.3000000
Class: > 15	0.6346154	0.4423077

The negative predictive value is displayed below, and class 15 showed a considerable increase. Instances are therefore less likely to be classified into class 15 if they are not truly of class 15.

Table 27: Neg Pred Value

	Pruned Rules	Decision Tree
Class: < 4	0.9927361	0.9927361
Class: 4	0.9926650	0.9853301
Class: 5	0.9665012	0.9751861
Class: 6	0.8997429	0.9228792
Class: 7	0.9320905	0.9214381
Class: 8	0.9706704	0.9678771
Class: 9	0.9205202	0.9263006
Class: 10	NA	0.9473684
Class: 11	0.8333333	0.8224044
Class: 12	0.9548969	0.9548969
Class: 13	NA	NA
Class: 14	0.8532338	NA
Class: 15	0.9752781	0.8257108
Class: > 15	0.8918919	0.9227799

The balanced accuracy is displayed below. This measure shows that classes 4, 8, 9, 10 and > 15 performed better in the pruned rule set. The decision tree performed better in classes 5, 6, and 7 and besides the classes that were not predicted, the rest of the classes seem to perform similarly.

Table 28: Balanced Accuracy

	Pruned Rules	Decision Tree
Class: < 4	0.6243910	0.6243910
Class: 4	0.6617886	0.5950860
Class: 5	0.6468841	0.6585315
Class: 6	0.6077586	0.6343391
Class: 7	0.6081847	0.6179949
Class: 8	0.6915502	0.6664700
Class: 9	0.6332444	0.6091539
Class: 10	NA	0.5409598
Class: 11	0.5715761	0.5654612
Class: 12	0.5078681	0.5078681
Class: 13	NA	NA
Class: 14	0.5283635	NA
Class: 15	0.5120521	0.5101442
Class: > 15	0.6276829	0.6191172

Covering Rule Induction Algorithm

I proceeded to investigate the covering rule induction algorithm. I decided to use the CN2 Induction Algorithm, which was developed by studying and learning from the ID3, AQ, and AQR algorithms.

The ID3, or Iterative Dichotomiser 3, algorithm is an old decision tree learner that iterates throughout a data set's set of previously unused features, and splits on the feature that minimizes the entropy the most. This algorithm can easily overfit noise in the data set, and therefore it requires pruning after induction. This algorithm typically terminates when either a leaf node is reached or when the data set's features have all been used. To generalize to new data, the resulting decision tree is simply traversed by the new observations' attributes and the class label is predicted as the leaf node's class label.

AQR induces decision rules from the training data by using the AQ algorithm. These rules are generated such that there is one rule per class, and the decision rules are formatted in an if then predict . In this format, represents a boolean combination of selectors, where a selector refers to a test to perform on a data set's attributes. By using the training data, the algorithm searches for an optimal complex to fit the training data. To do this, it uses an heuristic function to generate a set of optimal complexes, in which the rules maximize and minimize the amount of positive and negative examples covered by a rule. From this list of complexes, the complex that covers the most positive examples is selected as the cover that covers the training data the best.

The algorithm specializes this list of complexes that gets chosen from, by checking how many of the instances are covered by a complex and removing them from the training set. To then extend this cover further, it specializes the existing cover to cover the remaining positive classes, as well as remove any negative classes, by adding more terms to the cover. This induction process is repeated until all positive classes are covered and negative classes are excluded. At this point the best complex is chosen, and the process is repeated for all classes within the data set. Due to the nature of this algorithm, it overfits the classes. To generalize to new data, the resulting set of rules is used and the new instances are labelled as the class associated with the rule that satisfied the instance. If this new instance satisifies more than one rule's cover, it is assigned to the rule that is covered by the most instances.

The CN2 algorithm retains the beam search functionality and the IF-THEN rule format of the AQR algorithm, and it draws from ID3 by adding a cut-off method for specializing the complexes that is similar to decision tree pruning, and also by broadening the search space of the algorithm by including negative examples in a similar way to a ID3's growing process. CN2 presents an ordered set of if then predict rules, in which the final rule in the set contains a default rule that predicts the most frequent class in the training instances. In this sense, when the rules are tried from top-bottom it will simply predict the most frequent class.

The CN2 algorithm starts by selecting a single class and searching for the best complex or complexes to fit this class's instances. It achieves this by a heuristic, which searches for a complex that covers the maximum number of instances for this class. This complex is found by performing a beam search of the complex space, and in CN2, the considered complexes are generated as specializations of the best complexes found up until this stage. These specializations are generated by either adding new conjunctions to a complex, or by removing disjunctive elements from the selectors of a complex. All these newly specialized complexes are evaluated, and if one is better than any of the previous best-found complexes, it is added to this set of best complexes. The size of this set of best complexes is fixed, and the worst performing complex is removed from this set of complexes. After finding this specialized complex, all the instances covered by it are removed from the data set and the process is repeated until all the instances are covered.

The induced CN2 is displayed below with a best rule set size of 5. I did not perform any cross validation to find this best rule set size, I chose it as it seemed to be a common choice among implementations.

Table 29: Extracted Rule Set

	Rule
1	IF Shell.weight is [0.67, Inf] and Shucked.weight is [0.497,0.992) and Viscera.weight is [0.254,0.507) THEN Rings is > 15; (supportSize=31; laplace=0.4667)
2	IF Length is [-Inf,0.322) and Gender is A THEN Rings is < 4 ; (supportSize=265; laplace=0.4516)
3	IF Diameter is [-Inf,0.253) and Length is [0.322,0.568) and Gender is I THEN Rings is 6; (supportSize=99; laplace=0.3894)
4	IF Length is [-Inf,0.322) and Diameter is [-Inf,0.253) THEN Rings is 4; (supportSize=1422; laplace=0.3447)
5	IF Viscera.weight is [0.507, Inf] and Shucked.weight is [0.992, Inf] THEN Rings is 11; (supportSize=18; laplace=0.3125)
6	IF Viscera.weight is [0.507, Inf] THEN Rings is 12; (supportSize=16; laplace=0.3)
7	IF Shell.weight is [0.67, Inf] and Whole.weight is [1.88, Inf] THEN Rings is 14; (supportSize=7; laplace=0.2857)
8	IF Gender is I and Shell.weight is [0.336,0.67) and Whole.weight is [0.943,1.88) and Viscera.weight is [-Inf,0.254) and Length is [0.568, Inf] THEN Rings is > 15; (supportSize=21; laplace=0.3429)
9	IF Diameter is [-Inf,0.253) THEN Rings is 6; (supportSize=31; laplace=0.2667)
10	IF Shell.weight is [0.336,0.67) and Length is [0.322,0.568) and Diameter is [0.452, Inf] and Gender is A THEN Rings is > 15; (supportSize=17; laplace=0.3871)
11	IF Shell.weight is [0.336,0.67) and Whole.weight is [-Inf,0.943) and Length is [0.568, Inf] THEN Rings is 14; (supportSize=13; laplace=0.4074)
12	IF Shell.weight is [0.336,0.67) and Length is [0.322,0.568) and Whole.weight is [0.943,1.88) and Shucked.weight is [-Inf,0.497) THEN Rings is 15; (supportSize=40; laplace=0.3889)
13	IF Shucked.weight is [0.497,0.992) and Diameter is [0.253,0.452) and Viscera.weight is [0.254,0.507) and Gender is A THEN Rings is 10; (supportSize=16; laplace=0.3667)
14	IF Shucked.weight is [0.497,0.992) and Length is [0.322,0.568) THEN Rings is 9; (supportSize=14; laplace=0.2857)
15	IF Whole.weight is [1.88, Inf] and Gender is A and Height is [-Inf,0.377) THEN Rings is 12; (supportSize=134; laplace=0.2568)
16	IF Length is [-Inf,0.322) THEN Rings is 6; (supportSize=6; laplace=0.25)
17	IF Gender is I and Shucked.weight is [0.497,0.992) and Viscera.weight is [0.254,0.507) and Diameter is [0.452, Inf] and Whole.weight is [0.943,1.88) THEN Rings is 15; (supportSize=41; laplace=0.2909)
18	IF Shell.weight is [0.336,0.67) and Shucked.weight is [-Inf,0.497) and Viscera.weight is [0.254,0.507) THEN Rings is > 15; (supportSize=161; laplace=0.2629)
19	IF Gender is I and Diameter is [0.253,0.452) and Whole.weight is [0.943,1.88) and Shucked.weight is [-Inf,0.497) and Viscera.weight is [-Inf,0.254) THEN Rings is > 15; (supportSize=10; laplace=0.3333)
20	IF Shucked.weight is [0.497,0.992) and Diameter is [0.253,0.452) and Gender is A THEN Rings is 10; (supportSize=21; laplace=0.2571)
21	IF Gender is I and Shucked.weight is [0.497,0.992) and Whole.weight is [0.943,1.88) THEN Rings is 12; (supportSize=11; laplace=0.24)
22	IF Shucked.weight is [0.497,0.992) and Shell.weight is [-Inf,0.336) and Viscera.weight is [-Inf,0.254) THEN Rings is 9; (supportSize=132; laplace=0.2534)
23	IF Gender is I and Length is [0.322,0.568) and Diameter is [0.253,0.452) THEN Rings is 7; (supportSize=1374; laplace=0.2378)
24	IF Gender is I and Whole.weight is [-Inf,0.943) THEN Rings is 9; (supportSize=66; laplace=0.2375)
25	IF Shell.weight is [0.336,0.67) and Diameter is [0.253,0.452) and Length is [0.568, Inf] THEN Rings is 14; (supportSize=20; laplace=0.2353)
26	IF Shell.weight is [0.336,0.67) and Shucked.weight is [-Inf,0.497) THEN Rings is > 15; (supportSize=208; laplace=0.2252)
27	IF Gender is I THEN Rings is 10; (supportSize=28; laplace=0.2143)
28	IF Whole.weight is [0.943,1.88) and Length is [0.322,0.568) and Diameter is [0.253,0.452) THEN Rings is > 15; (supportSize=75; laplace=0.3146)

	Rule
29	IF Viscera.weight is [0.254,0.507) and Whole.weight is [-Inf,0.943) and Length is [0.568, Inf] THEN Rings is 9; (supportSize=4; laplace=0.2222)
30	IF Viscera.weight is [0.254,0.507) and Diameter is [0.253,0.452) THEN Rings is 14; (supportSize=8; laplace=0.2273)
31	IF Whole.weight is [0.943,1.88) and Length is [0.322,0.568) and Viscera.weight is [-Inf,0.254) THEN Rings is 14; (supportSize=13; laplace=0.2222)
32	IF Whole.weight is [0.943,1.88) and Diameter is [0.253,0.452) THEN Rings is 13; (supportSize=47; laplace=0.2131)
33	IF Viscera.weight is [0.254,0.507) and Shucked.weight is [-Inf,0.497) and Length is [0.568, Inf] THEN Rings is 14; (supportSize=119; laplace=0.2105)
34	IF Shucked.weight is [0.497,0.992) and Shell.weight is [-Inf,0.336) THEN Rings is 9; (supportSize=190; laplace=0.2059)
35	IF Whole.weight is [0.943,1.88) and Shucked.weight is [-Inf,0.497) and Length is [0.568, Inf] THEN Rings is 15; (supportSize=240; laplace=0.2008)
36	IF Diameter is [0.452, Inf] and Length is [0.322,0.568) THEN Rings is 11; (supportSize=26; laplace=0.2)
37	IF Viscera.weight is [0.254,0.507) and Height is [-Inf,0.377) THEN Rings is 11; (supportSize=957; laplace=0.1864)
38	IF Length is [0.568, Inf] and Diameter is [0.253,0.452) THEN Rings is 8; (supportSize=103; laplace=0.2137)
39	IF Whole.weight is [0.943,1.88) and Shell.weight is [0.336,0.67) THEN Rings is 14; (supportSize=107; laplace=0.1818)
40	IF Length is [0.568, Inf] and Whole.weight is [-Inf,0.943) THEN Rings is 13; (supportSize=94; laplace=0.1574)
41	IF Height is [0.377,0.753) THEN Rings is 10; (supportSize=1; laplace=0.1333)
42	IF Length is [0.568, Inf] THEN Rings is 12; (supportSize=1; laplace=0.1333)
43	IF Height is [0.753, Inf] THEN Rings is 8; (supportSize=1; laplace=0.1333)
44	IF Gender is A THEN Rings is 15; (supportSize=1517; laplace=0.1228)

The first thing I noticed about the extracted rules is the simplicity of them when compared to the previous rules extracted. For instance, note rules 2, 5, and 7 in which the covers are constructed from the conjunction of two statements. Also note rule 6 and 27 which are constructed from only only a single statement. Upon closer inspection of the library I used, I discovered that this function applies automatic pruning. I could not implement my own pruning process, and due to a lack of time I will unfortunately not be able to complete question 5 and 6 of the assignment in their entirety as this rule set is already pruned. I will however attempt to do as much of both of these questions as possible.

The second thing I saw was the final and default rule received a significant amount of supporting instances, as denoted by supportSize. These are the instances that pass through the rule set and are assigned as the most frequently occurring class. Once again these rules are substantially less complicated than the decision tree rules, and I believe the most complex rule is Rule 19, which is constructed from 5 IF statements. The majority of these rules seem to be constructed from 3 statements, and even though this model is already pruned, this algorithm requires more rules than the unpruned and pruned rule based classifiers previously discussed. There is a variable named laplace at the end of this rule, and it provides the rules' Laplace estimate of accuracy. I then proceeded to quantify the performance of these rules in terms of coverage and accuracy.

Table 30: Covering Rule Performance

	Coverage	Accuracy
Rule_1	0.0040129	0.6451613
Rule_2	0.0343042	0.4716981
Rule_3	0.0128155	0.4343434
Rule_4	0.1840777	0.3473980
Rule_5	0.0023301	0.5000000
Rule_6	0.0020712	0.5000000
Rule_7	0.0009061	0.7142857
Rule_8	0.0027184	0.5238095
Rule_9	0.0040129	0.3548387
Rule_10	0.0022006	0.6470588
Rule_11	0.0016828	0.7692308
Rule_12	0.0051780	0.5000000
Rule_13	0.0020712	0.6250000
Rule_14	0.0018123	0.5000000
Rule_15	0.0173463	0.2761194
Rule_16	0.0007767	0.6666667
Rule_17	0.0053074	0.3658537
Rule_18	0.0208414	0.2795031
Rule_19	0.0012945	0.7000000
Rule_20	0.0027184	0.3809524
Rule_21	0.0014239	0.4545455
Rule_22	0.0170874	0.2727273
Rule_23	0.1778641	0.2394469
Rule_24	0.0085437	0.2727273
Rule_25	0.0025890	0.3500000
Rule_26	0.0269256	0.2355769
Rule_27	0.0036246	0.2857143
Rule_28	0.0097087	0.3600000
Rule_29	0.0005178	0.7500000
Rule_30	0.0010356	0.5000000
Rule_31	0.0016828	0.3846154
Rule_32	0.0060841	0.2553191
Rule_33	0.0154045	0.2268908
Rule_34	0.0245955	0.2157895
Rule_35	0.0310680	0.2083333
Rule_36	0.0033657	0.2692308
Rule_37	0.1238835	0.1880878
Rule_38	0.0133333	0.2330097
Rule_39	0.0138511	0.1962617
Rule_40	0.0121683	0.1702128
Rule_41	0.0001294	1.0000000
Rule_42	0.0001294	1.0000000
Rule_43	0.0001294	1.0000000
Rule_44	0.1963754	0.1232696

The CN2 algorithm is quantified in terms of coverage and accuracy as seen above. There are quite a lot of rules so it is hard to investigate. Let's take a look at all rules that had accuracies above 50%.

Table 31: Extracted Rules: Accuracies > 50%

	Coverage	Accuracy
Rule_41	0.0001294	1.0000000
Rule_42	0.0001294	1.0000000
Rule_43	0.0001294	1.0000000
Rule_11	0.0016828	0.7692308
Rule_29	0.0005178	0.7500000
Rule_7	0.0009061	0.7142857
Rule_19	0.0012945	0.7000000
Rule_16	0.0007767	0.6666667
Rule_10	0.0022006	0.6470588
Rule_1	0.0040129	0.6451613
Rule_13	0.0020712	0.6250000
Rule_8	0.0027184	0.5238095

Unfortunately, all the rules with accuracies above 50% have very low coverages. The rule with the highest coverage among these ten rules is rule 1, which only has 31 supporting instances and an accuracy of 64.5%. Let's investigate the rules that had coverages exceeding 10%.

Table 32: Extracted Rules: Coverage > 10%

	Coverage	Accuracy
Rule_44	0.1963754	0.1232696
Rule_4	0.1840777	0.3473980
Rule_23	0.1778641	0.2394469
Rule_37	0.1238835	0.1880878

The coverages of all rules exceeding 10% are displayed above. I noted that the default rule had the highest coverage in the data set, which is somewhat troubling. I also noted that rule 4, with a class label of 4, had the second highest coverage and a higher accuracy than the other three rules in this table. I then proceeded to investigate the generalization performance of these extracted rules.

Table 33: Accuracy

acc
0.1809409

The accuracy is displayed above, and it can be seen to be very poor, in fact it has a lower accuracy than the previous two methods.

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	F1	Balanced Accuracy
Class: < 4	0.0000000	0.9963190	0.0000000	0.9830508	NaN	0.4981595
Class: > 15	0.2539683	0.9530026	0.3076923	0.9395109	0.2782609	0.6034854
Class: 10	0.5454545	0.8533007	0.0476190	0.9928876	0.0875912	0.6993776
Class: 11	0.2666667	0.9046961	0.2886598	0.8948087	0.2772277	0.5856814
Class: 12	0.0714286	0.9361963	0.0188679	0.9832474	0.0298507	0.5038124

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	F1	Balanced Accuracy
Class: 13	0.0000000	0.9501247	0.0000000	0.9657795	NaN	0.4750623
Class: 14	0.0857143	0.9722922	0.1200000	0.9601990	0.1000000	0.5290032
Class: 15	0.0177778	0.9735099	0.2000000	0.7268232	0.0326531	0.4956439
Class: 4	0.1875000	0.9974392	0.8181818	0.9523227	0.3050847	0.5924696
Class: 6	0.3750000	0.9415347	0.0588235	0.9935733	0.1016949	0.6582674
Class: 7	0.2712766	0.9578783	0.6538462	0.8175766	0.3834586	0.6145775
Class: 8	0.2777778	0.8668311	0.0442478	0.9818436	0.0763359	0.5723044
Class: 9	0.3287671	0.8505291	0.1751825	0.9291908	0.2285714	0.5896481
Class: 5	NA	0.9722557	NA	NA	NA	NA

Further performance measures are displayed above, and when comparing these measures to those illustrated on page 20 and 21, this algorithm can be seen to perform worse than both extracted tree rule sets in almost every possible way. Class 4, 5 and 13 were never classified correctly as seen in the sensitivity. Class 4 and 7 performed slightly better in terms of specificity than the previous two models, but as their sensitivities have decreased, the model just classified less instances as belonging to these two classes and this increase in specificity is not indicative of a better performing model. Both the PPV and NPV shows that Class > 15 is performing significantly worse than the extracted tree rule sets. The balanced accuracies for each rule set is shown below. We can see that class 10 and 6 achieved higher balanced accuracies than the decision trees, and class < 4 performed significantly worse.

Table 35: Balanced Accuracy

	Pruned Rules	Decision Tree
Class: < 4	0.6243910	0.6243910
Class: 4	0.6617886	0.5950860
Class: 5	0.6468841	0.6585315
Class: 6	0.6077586	0.6343391
Class: 7	0.6081847	0.6179949
Class: 8	0.6915502	0.6664700
Class: 9	0.6332444	0.6091539
Class: 10	NA	0.5409598
Class: 11	0.5715761	0.5654612
Class: 12	0.5078681	0.5078681
Class: 13	NA	NA
Class: 14	0.5283635	NA
Class: 15	0.5120521	0.5101442
Class: > 15	0.6276829	0.6191172

Table 36: Balanced accuracy

	Covering Algorithm
Class: 10	0.6993776
Class: 6	0.6582674
Class: 7	0.6145775
Class: > 15	0.6034854
Class: 4	0.5924696
Class: 9	0.5896481
Class: 11	0.5856814
Class: 8	0.5723044
Class: 14	0.5290032

	Covering Algorithm
Class: 12	0.5038124
Class: < 4	0.4981595
Class: 15	0.4956439
Class: 13	0.4750623
Class: 5	NA

Rule Association

I proceeded to investigate the rule association algorithm. I selected the apriori algorithm. This algorithm generates rules from a data set by using frequent itemsets and the apriori property. This apriori property states that the subset of any itemset that has a supporting value below a certain threshold can be discarded, as any subset of this itemset will also have a supporting value below the threshold. If an itemset has a supporting value that exceeds this threshold, it is considered a frequent itemset and gets extended by candidate generation.

The process of this algorithm can be explained as follow. First, the data is scanned to find all frequent sets with k items (for the first iteration k = 1). This is called a k-itemset, and after generation it is pruned according to the minimum specified support. All remaining instances are frequent itemsets. By using these frequent sets, we can generate frequent sets with k+1 candidate items by self-joining the frequent sets. These new frequent sets are selected as candidates, and are joined into the existing k-itemset. The frequent sets are then pruned and the candidate generation process is repeated until the self-joining process leaves the set of frequent itemsets empty, at which point the rule association algorithm is terminated.

I then proceeded to implement the algorithm and based on previous applications I selected a maximum length of 10 items for the rules extracted. Similarly, I chose an accuracy of 60% as this value seemed to be what was considered good in the decision trees and covering algorithms. Finally, I used a support value of 0.001 as the smoting process ensured that the coverage of the data set is quite low. This associated a set of 136 rules, and the first five rules are displayed below.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 7
##
## set item appearances ...[14 item(s)] done [0.00s].
## set transactions ...[37 item(s), 7725 transaction(s)] done [0.00s].
## sorting and recoding items ... [37 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 done [0.01s].
## writing ... [136 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

## set of 136 rules
```

```
##
## rule length distribution (lhs + rhs):sizes
##  4  5  6  7  8  9
##  3 16 43 46 24  4
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.000   6.000   7.000   6.618   7.000   9.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.001036   Min.   :0.6000   Min.   :0.001294   Min.   : 8.397
##  1st Qu.:0.001165   1st Qu.:0.6239   1st Qu.:0.001812   1st Qu.: 8.735
##  Median :0.001424   Median :0.6667   Median :0.002071   Median : 9.330
##  Mean   :0.001514   Mean   :0.6739   Mean   :0.002283   Mean   : 9.434
##  3rd Qu.:0.001812   3rd Qu.:0.7143   3rd Qu.:0.002718   3rd Qu.: 9.996
##  Max.   :0.003107   Max.   :0.8000   Max.   :0.005049   Max.   :11.196
##      count
##  Min.   : 8.0
##  1st Qu.: 9.0
##  Median :11.0
##  Mean   :11.7
##  3rd Qu.:14.0
##  Max.   :24.0
##
## mining info:
##         data ntransactions support confidence
##  train_smote          7725   0.001        0.6
```

It can be seen that the majority of rules 6 - 7 statements long.

I then proceeded to investigate the top five rules in terms of support. The highest achieved support is still quite low, at 0.003106796.

```
##     lhs                                rhs           support confidence    coverage     lift count
## [1] {Length=[0.42,0.576),
##      Diameter=[0.323,0.452),
##      Whole.weight=[0.353,0.974),
##      Shell.weight=[0.0015,0.104)}  => {Rings=6} 0.003106796  0.6153846 0.005048544 8.627670    24
## [2] {Gender=I,
##      Length=[0.42,0.576),
##      Diameter=[0.323,0.452),
##      Whole.weight=[0.353,0.974),
##      Shell.weight=[0.0015,0.104)}  => {Rings=6} 0.003106796  0.6315789 0.004919094 8.854714    24
## [3] {Gender=I,
##      Length=[0.42,0.576),
##      Diameter=[0.323,0.452),
##      Whole.weight=[0.353,0.974),
##      Viscera.weight=[0.073,0.207),
##      Shell.weight=[0.0015,0.104)}  => {Rings=6} 0.002847896  0.6111111 0.004660194 8.567756    22
## [4] {Gender=I,
##      Length=[0.42,0.576),
##      Diameter=[0.323,0.452),
##      Whole.weight=[0.353,0.974),
##      Shucked.weight=[0.152,0.4),
```

```
##      Shell.weight=[0.0015,0.104)}  => {Rings=6} 0.002847896  0.6111111 0.004660194 8.567756    22
## [5] {Gender=I,
##      Length=[0.42,0.576),
##      Diameter=[0.323,0.452),
##      Whole.weight=[0.353,0.974),
##      Shucked.weight=[0.152,0.4),
##      Viscera.weight=[0.073,0.207),
##      Shell.weight=[0.0015,0.104)}  => {Rings=6} 0.002847896  0.6111111 0.004660194 8.567756    22
```

I then proceeded to investigate the top five rules in terms of accuracy. The highest achieved accuracy was 80% as seen below.

```
##     lhs                              rhs              support confidence    coverage     lift count
## [1] {Gender=I,
##      Whole.weight=[0.974,2.83],
##      Shucked.weight=[0.152,0.4)} => {Rings=> 15} 0.001035599        0.8 0.001294498 11.19565     8
## [2] {Gender=I,
##      Whole.weight=[0.974,2.83],
##      Shucked.weight=[0.152,0.4),
##      Shell.weight=[0.288,1]}     => {Rings=> 15} 0.001035599        0.8 0.001294498 11.19565     8
## [3] {Gender=I,
##      Height=[0.155,1.13],
##      Whole.weight=[0.974,2.83],
##      Shucked.weight=[0.152,0.4)} => {Rings=> 15} 0.001035599        0.8 0.001294498 11.19565     8
## [4] {Length=[0.075,0.42),
##      Diameter=[0.323,0.452),
##      Height=[0,0.105),
##      Whole.weight=[0.353,0.974)} => {Rings=7}    0.001035599        0.8 0.001294498 11.19565     8
## [5] {Gender=I,
##      Height=[0.155,1.13],
##      Whole.weight=[0.974,2.83],
##      Shucked.weight=[0.152,0.4),
##      Shell.weight=[0.288,1]}     => {Rings=> 15} 0.001035599        0.8 0.001294498 11.19565     8
```

References

https://rdrr.io/cran/rpart/f/inst/doc/longintro.pdf

https://towardsdatascience.com/decision-trees-d07e0f420175

https://www.tc.columbia.edu/elda/blog/content/receiver-operating-characteristic-roc-area-under-the-curve-auc/

Details about Performance Measures: Post Block Assignment 2, Module: Data Science 874, 18620426, Francois van Zyl, 27 March 2020, Stellenbosch University

Class Slides for Topics 3 and 4, Data Analytics 874, Stellenbosch University, 2021

P. CLARK AND T. NIBLETT, The CN2 Induction Algorithm, October 25, 1988

https://www.upgrad.com/blog/apriori-algorithm/