FIT3152 Data Analytics

Assignment 2

Zoe Yow Cui Yi | 33214476

Table of Contents

Questions

Question 1:

The dataset contains 5000 rows and 21 columns, where 20 of the columns (A02-A30) were remote sensing data containing various optical and radar features. This information was used to determine if the type of crop that is being grown on farmland is oats or not and the results were stored in the "Class" column. The data types of columns "A02" to "A30" are numerical while the "Class" column is integer data type.

The proportion of "Oats" (class = 1) to "Other" (class = 0) crops is shown in Figure 1. Observing the plot, "Other" crops have a proportion that is more than 0.8, while "Oats" crops have a proportion of less than 0.2. This shows that the dataset proportion is imbalanced with "Other" crops having more rows of data than "Oats" crops.

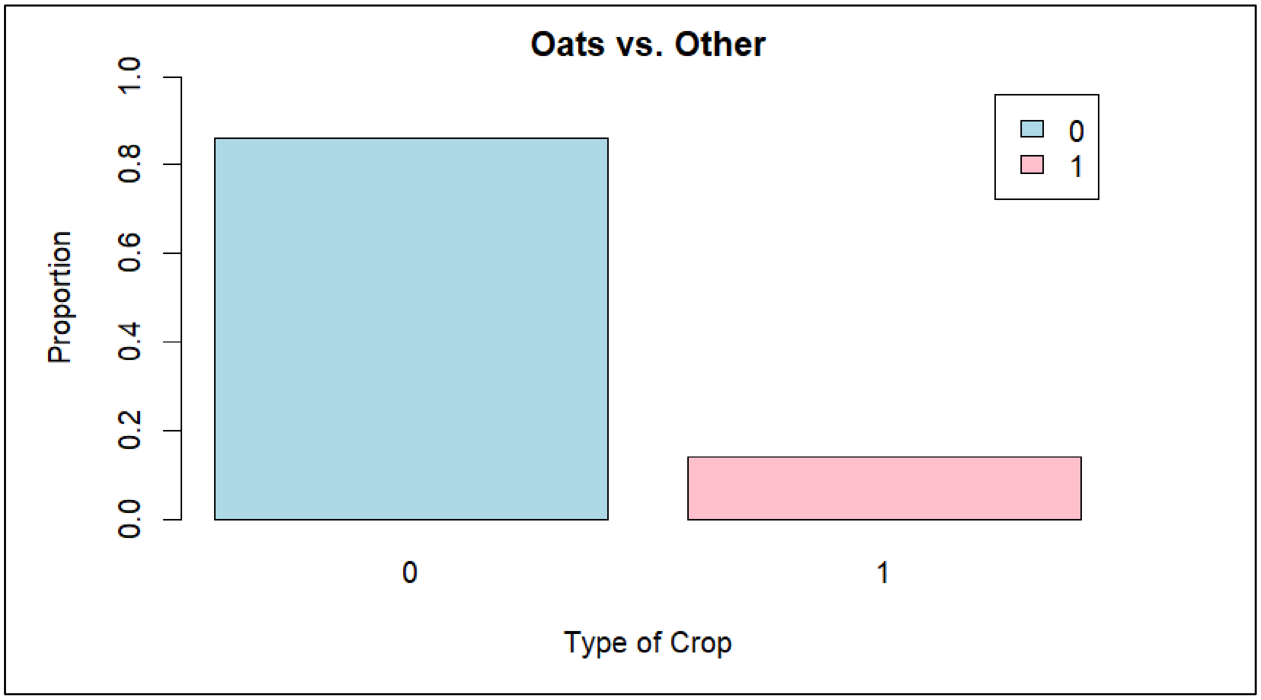
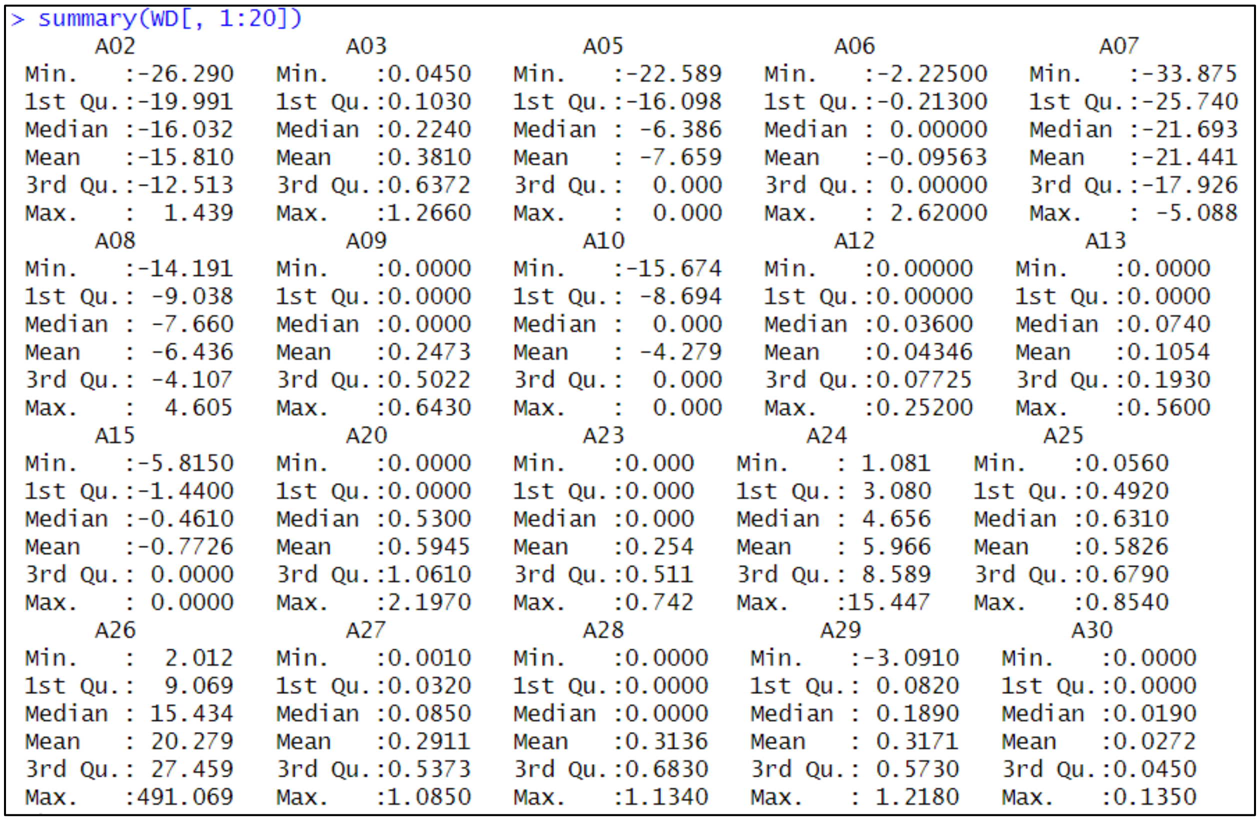


Figure 1: Bar Chart of the Proportion of "Oats" (class = 1), to "Other" (class = 0) crops

To obtain the distribution of numerical attributes shown below in Figure 2, the "summary" function was used. The descriptions obtained include the minimum, 1st quartile, median, mean, 3rd quartile, and maximum. All columns except column "A26" and "A24" have values less than 5. Moreover, the standard deviation of all numerical attributes was calculated and the standard deviation of column "A26" is significantly larger than other numerical attributes. This shows that column "A26" have a more spread-out data and might potentially be a significant factor in predicting

"Oats" (class = 1), or "Other" (class = 0). For now, there are no attributes I need to consider omitting from my analysis as it is too early in the analysis to make such a decision.



```
> summary(WD[, 1:20])
      A02              A03               A05              A06                 A07
 Min.   :-26.290   Min.   :0.0450   Min.   :-22.589   Min.   :-2.22500   Min.   :-33.875
 1st Qu.:-19.991   1st Qu.:0.1030   1st Qu.:-16.098   1st Qu.:-0.21300   1st Qu.:-25.740
 Median :-16.032   Median :0.2240   Median : -6.386   Median : 0.00000   Median :-21.693
 Mean   :-15.810   Mean   :0.3810   Mean   : -7.659   Mean   :-0.09563   Mean   :-21.441
 3rd Qu.:-12.513   3rd Qu.:0.6372   3rd Qu.:  0.000   3rd Qu.: 0.00000   3rd Qu.:-17.926
 Max.   :  1.439   Max.   :1.2660   Max.   :  0.000   Max.   : 2.62000   Max.   : -5.088
      A08              A09               A10              A12                A13
 Min.   :-14.191   Min.   :0.0000   Min.   :-15.674   Min.   :0.00000   Min.   :0.0000
 1st Qu.: -9.038   1st Qu.:0.0000   1st Qu.: -8.694   1st Qu.:0.00000   1st Qu.:0.0000
 Median : -7.660   Median :0.0000   Median :  0.000   Median :0.03600   Median :0.0740
 Mean   : -6.436   Mean   :0.2473   Mean   : -4.279   Mean   :0.04346   Mean   :0.1054
 3rd Qu.: -4.107   3rd Qu.:0.5022   3rd Qu.:  0.000   3rd Qu.:0.07725   3rd Qu.:0.1930
 Max.   :  4.605   Max.   :0.6430   Max.   :  0.000   Max.   :0.25200   Max.   :0.5600
      A15              A20               A23             A24              A25
 Min.   :-5.8150   Min.   :0.0000   Min.   :0.000   Min.   : 1.081   Min.   :0.0560
 1st Qu.:-1.4400   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.: 3.080   1st Qu.:0.4920
 Median :-0.4610   Median :0.5300   Median :0.000   Median : 4.656   Median :0.6310
 Mean   :-0.7726   Mean   :0.5945   Mean   :0.254   Mean   : 5.966   Mean   :0.5826
 3rd Qu.: 0.0000   3rd Qu.:1.0610   3rd Qu.:0.511   3rd Qu.: 8.589   3rd Qu.:0.6790
 Max.   : 0.0000   Max.   :2.1970   Max.   :0.742   Max.   :15.447   Max.   :0.8540
      A26              A27               A28              A29               A30
 Min.   :  2.012   Min.   :0.0010   Min.   :0.0000   Min.   :-3.0910   Min.   :0.0000
 1st Qu.:  9.069   1st Qu.:0.0320   1st Qu.:0.0000   1st Qu.: 0.0820   1st Qu.:0.0000
 Median : 15.434   Median :0.0850   Median :0.0000   Median : 0.1890   Median :0.0190
 Mean   : 20.279   Mean   :0.2911   Mean   :0.3136   Mean   : 0.3171   Mean   :0.0272
 3rd Qu.: 27.459   3rd Qu.:0.5373   3rd Qu.:0.6830   3rd Qu.: 0.5730   3rd Qu.:0.0450
 Max.   :491.069   Max.   :1.0850   Max.   :1.1340   Max.   : 1.2180   Max.   :0.1350
```

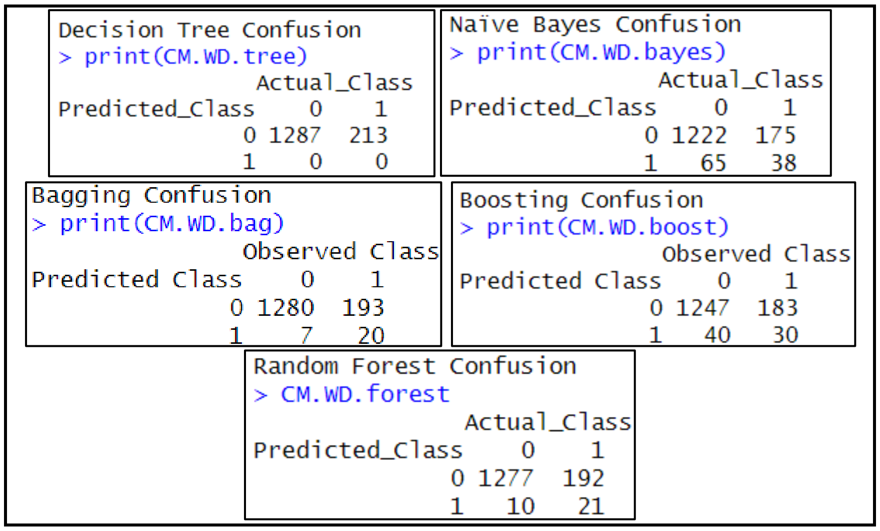
Figure 2: Distribution of Numerical Attributes

Question 2:

There are no missing values in this dataset, so no further handling of missing values was needed. Therefore, the only pre-processing required for this dataset was to convert the Class variable into data type factor from data type integer. This conversion is important for treating the Class variable as the target variable in classification tasks.

Question 3 and Question 4 and Question 5:

Answers are in R-studio. The R-code can be found in the Appendix.



```
Decision Tree Confusion
> print(CM.WD.tree)
               Actual_Class
Predicted_Class    0    1
              0 1287  213
              1    0    0
```

```
Naïve Bayes Confusion
> print(CM.WD.bayes)
               Actual_Class
Predicted_Class    0    1
              0 1222  175
              1   65   38
```

```
Bagging Confusion
> print(CM.WD.bag)
               Observed Class
Predicted Class    0    1
              0 1280  193
              1    7   20
```

```
Boosting Confusion
> print(CM.WD.boost)
               Observed Class
Predicted Class    0    1
              0 1247  183
              1   40   30
```

```
Random Forest Confusion
> CM.WD.forest
               Actual_Class
Predicted_Class    0    1
              0 1277  192
              1   10   21
```

Figure 3: Confusion Matrix of All Classifiers

Question 6:

The ROC Graph containing all classification models are shown in Figure 3 below.

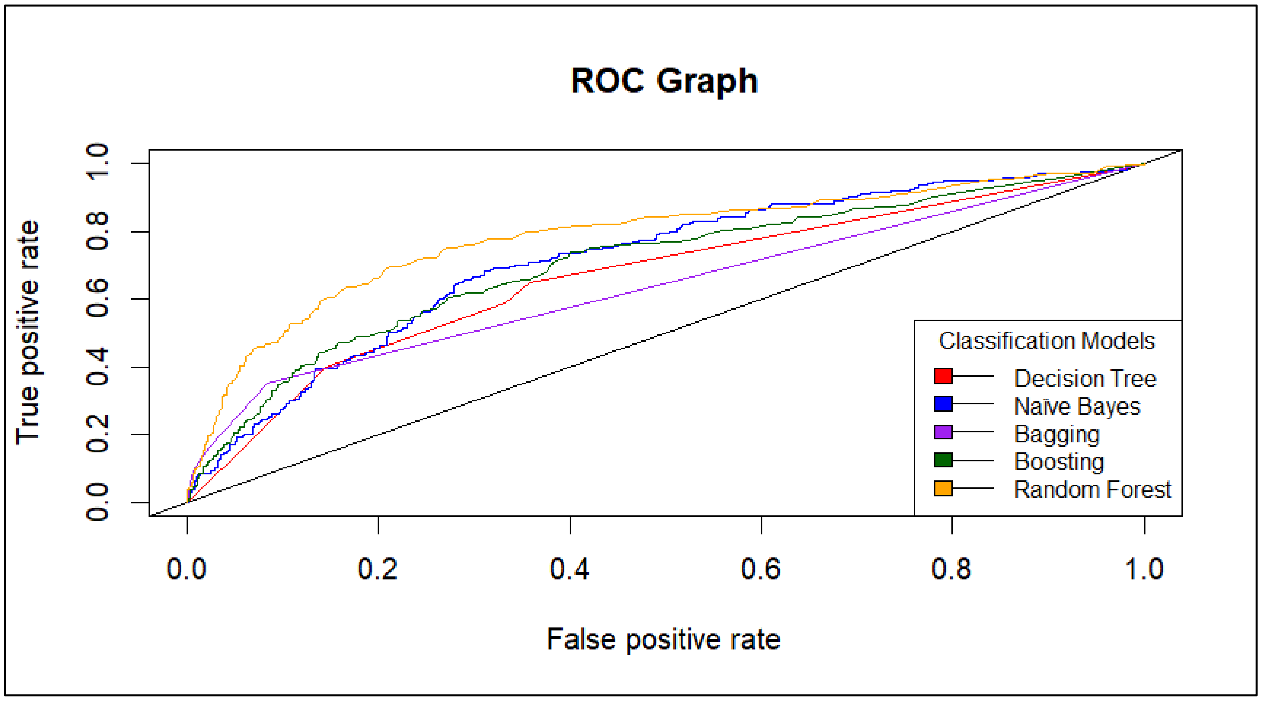


Figure 4: ROC Graph for All Classification Models

The Area Under the Curve (AUC) value for all classifiers:

Classifier	**AUC value**
Decision Tree	0.6647333
Naïve Bayes	0.7156104

Bagging	0.6375437
Boosting	0.7048017
Random Forest	0.7843951

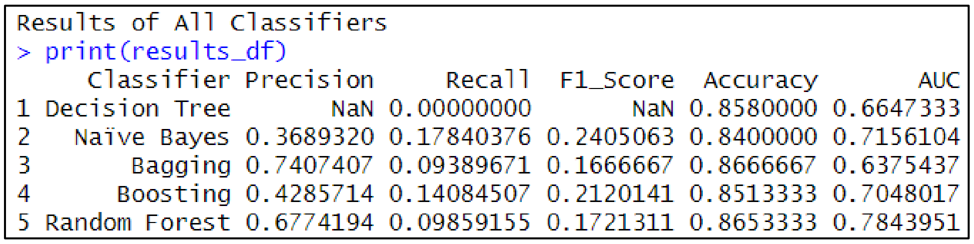
Question 7:

The prediction and performance results for all classifiers — including precision, recall, F1-score, accuracy, and AUC — are shown in Figure 5. Based on these results, all classifiers struggle with recall and F1-score, particularly in predicting the “Other” class (class = 0). This suggests that the models are better at identifying “Oats” (class = 1), but do not balance the prediction of both classes effectively.

Among the classifiers evaluated, the Random Forest model stands out as the most balanced and effective overall. While its recall (0.0986) and F1-score (0.1721) are relatively low, it achieves the highest AUC among all models, indicating a strong ability to distinguish between the two classes—“Oats” and “Other”—across all classification thresholds. This high AUC value suggests that Random Forest maintains good overall discriminatory power even if its performance on one class (particularly the minority class) is not optimal.

In comparison, the Bagging model shows the highest precision and accuracy, which at first glance might suggest it performs better. However, its lower recall, F1-score, and AUC indicate that it may be overfitting to the majority class and failing to identify “Oats” (class = 1) effectively. This trade-off is important because high precision alone is not sufficient in imbalanced datasets where recall and the ability to detect both classes are critical.

Therefore, despite its slightly lower precision and accuracy, the Random Forest model is considered the best overall due to its superior balance between metrics and stronger generalization ability as reflected in the AUC score.



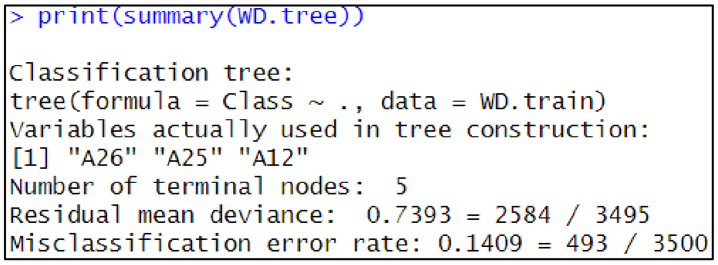
```
Results of All Classifiers
> print(results_df)
     Classifier Precision     Recall  F1_Score  Accuracy       AUC
1 Decision Tree       NaN 0.00000000       NaN 0.8580000 0.6647333
2   Naïve Bayes 0.3689320 0.17840376 0.2405063 0.8400000 0.7156104
3       Bagging 0.7407407 0.09389671 0.1666667 0.8666667 0.6375437
4      Boosting 0.4285714 0.14084507 0.2120141 0.8513333 0.7048017
5 Random Forest 0.6774194 0.09859155 0.1721311 0.8653333 0.7843951
```

Figure 5: Table of Results for All Classification Models

Investigative Tasks

Question 8:

Using a decision tree, the most important attributes used in predicting “Oats” (class = 1), or “Other” (class = 0) are A26, A25, and A12. This is obtained from Figure 6 below.

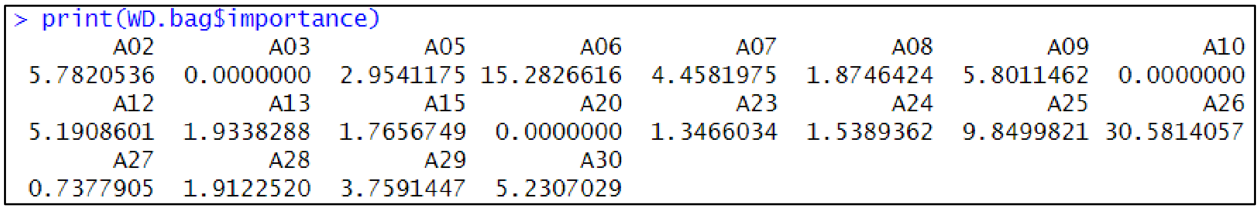


```
> print(summary(WD.tree))

Classification tree:
tree(formula = Class ~ ., data = WD.train)
Variables actually used in tree construction:
[1] "A26" "A25" "A12"
Number of terminal nodes:  5
Residual mean deviance:  0.7393 = 2584 / 3495
Misclassification error rate: 0.1409 = 493 / 3500
```

Figure 6: Most Important Attributes using Decision Tree

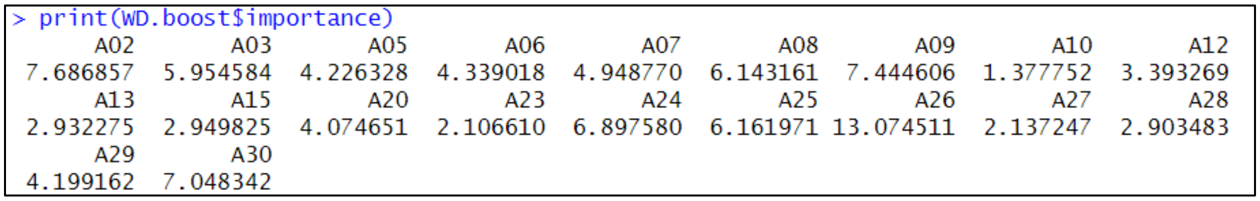
For Bagging, observing Figure 7 below, the most important attributes used in predicting “Oats” (class = 1), or “Other” (class = 0) are A26, A06, A25, and A09. Moreover, the least important attributes can be said to be A03, A20, A10, and A27.



```
> print(WD.bag$importance)
       A02        A03        A05        A06        A07        A08        A09        A10
 5.7820536  0.0000000  2.9541175 15.2826616  4.4581975  1.8746424  5.8011462  0.0000000
       A12        A13        A15        A20        A23        A24        A25        A26
 5.1908601  1.9338288  1.7656749  0.0000000  1.3466034  1.5389362  9.8499821 30.5814057
       A27        A28        A29        A30
 0.7377905  1.9122520  3.7591447  5.2307029
```

Figure 7: Attributes’ Importance using Bagging

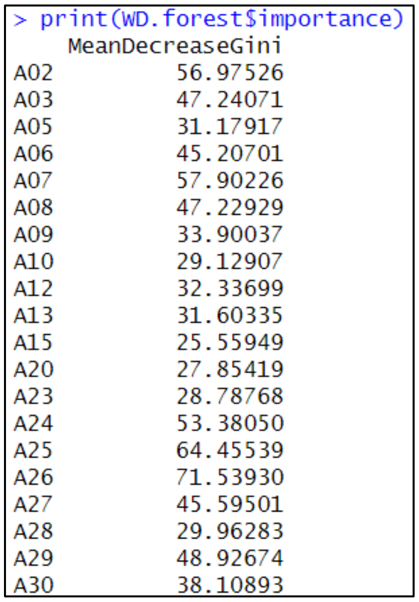
For boosting, looking at Figure 8 below, the most important attributes used in predicting “Oats” (class = 1), or “Other” (class = 0) are A26, A02, A09, A30, and A24. Also, the least important attributes are A10, A23, and A27.



```
> print(WD.boost$importance)
      A02       A03       A05       A06       A07       A08       A09       A10       A12
 7.686857  5.954584  4.226328  4.339018  4.948770  6.143161  7.444606  1.377752  3.393269
      A13       A15       A20       A23       A24       A25       A26       A27       A28
 2.932275  2.949825  4.074651  2.106610  6.897580  6.161971 13.074511  2.137247  2.903483
      A29       A30
 4.199162  7.048342
```

Figure 8: Attributes’ Importance using Boosting

Using random forest, Figure 9 shows that the most important attributes used in predicting “Oats” (class = 1), or “Other” (class = 0) are A26, A25, A07, A02, and A24. Additionally, the least important attributes are A15, A23, A20, A10, and A28



```
> print(WD.forest$importance)
    MeanDecreaseGini
A02         56.97526
A03         47.24071
A05         31.17917
A06         45.20701
A07         57.90226
A08         47.22929
A09         33.90037
A10         29.12907
A12         32.33699
A13         31.60335
A15         25.55949
A20         27.85419
A23         28.78768
A24         53.38050
A25         64.45539
A26         71.53930
A27         45.59501
A28         29.96283
A29         48.92674
A30         38.10893
```

Figure 9: Attributes' Importance using Random Forest

Overall, the most important attributes for predicting the class label are A26 and A25. This conclusion is based on the attribute importance shown in Figures 6 to 9. A26 and A25 consistently appear as the top predictors across all models, indicating their strong relevance in distinguishing between the "Oats" (class = 1) and "Other" (class = 0) categories. Although A12 is highlighted as important in the decision tree model, it does not rank in the top for the other classifiers, suggesting its influence might be model-specific and less reliable overall.

On the other hand, attributes like A23, A20, A10, A28, and A15 could potentially be removed from the dataset without significantly affecting model performance. These attributes are not used by the decision tree as shown in Figure 5 and show very low importance value in the other classifiers, which suggests they contribute little to the prediction task. Removing these less influential attributes can simplify the model and reduce potential noise in the data.

Question 9:

The performance differences between the classifiers—Decision Tree, Naïve Bayes, Bagging, Boosting, and Random Forest—can be explained by the nature of each model and how well they handle the specific characteristics of your dataset, such as class imbalance and potential feature interactions.

The Decision Tree model achieved a high accuracy of 0.858, but both its recall and F1-score are 0, meaning it failed to correctly identify any instances of the minority class ("Oats"). This is a common issue when using decision trees on imbalanced datasets, as they tend to overfit the majority class and fail to generalize to underrepresented classes. Despite the deceptively high accuracy, the model's inability to detect the minority class renders it ineffective in this context.

Naïve Bayes, with a precision of 0.3689 and a recall of 0.1780, performs better in terms of balance between the two classes. Its performance is modest due to its underlying assumption that all features are conditionally independent—an assumption that rarely holds in real-world data. However, it handles imbalanced data better than decision trees by computing class probabilities, leading to a more balanced but still limited performance.

Bagging improved precision substantially to 0.7407, but its recall dropped to 0.0939, resulting in a low F1-score. Bagging, which builds multiple trees on different random subsets of data, is designed to reduce model variance and overfitting. However, it does not explicitly address class imbalance. As a result, while it correctly predicts many of the majority class instances, it struggles with detecting the minority class.

Boosting performed slightly better in terms of F1-score than Bagging, with a recall of 0.1408 and an F1 of 0.2121. Boosting sequentially focuses on misclassified instances, allowing the model to gradually improve its handling of hard-to-classify cases. This approach helps in improving recall and better identifying minority class samples. However, it is more sensitive to noise and requires careful tuning to avoid overfitting.

Among all models, Random Forest stands out as the most balanced classifier. With a precision of 0.6774 and the highest AUC (0.7844), it combines the strengths of bagging and feature randomness to reduce overfitting and capture complex interactions. While its recall is still low at 0.0986, its superior AUC and consistent performance across metrics indicate strong discriminatory power and robustness. This makes Random Forest the best overall model for this dataset, especially when the goal is to maintain a balance between correctly predicting both classes while minimizing overfitting.

In summary, models like Boosting and Random Forest, which are ensemble methods, tend to outperform simpler models on complex, imbalanced data due to their ability to generalize and focus on difficult-to-predict samples. Simpler models like Decision Tree and Naïve Bayes, while easier to interpret and faster to train, are less capable of managing class imbalance and feature interactions effectively.

Question 10:

I decided to build a decision tree and it is shown in Figure 10 below. It is simple enough for a person to manually classify whether a crop is "Oats" (class = 1) or "Other" (class = 0). This tree has only three terminal nodes, making it easy to follow and interpret without the need for computational tools, all of which classify instances as "Other." This is likely due to the class imbalance in the training data — there are significantly more rows labeled "Other" than "Oats." As a result, the more complex tree lacks sufficient examples of the "Oats" class to effectively learn patterns for predicting it, leading to biased predictions toward the majority class.

The factors that were important in my decision were that this simple decision tree should have high accuracy despite its simplicity. Moreover, it should include the most important and relevant attributes identified across previous classifiers in Question 8, to ensure it captures key patterns in the data while remaining easy to interpret. In designing this model, I intentionally prioritized simplicity and interpretability over complexity, accepting a possible tradeoff in accuracy to create a model that could be applied manually. This balance allows the model to remain understandable while still performing adequately in predicting "Oats" or "Other".

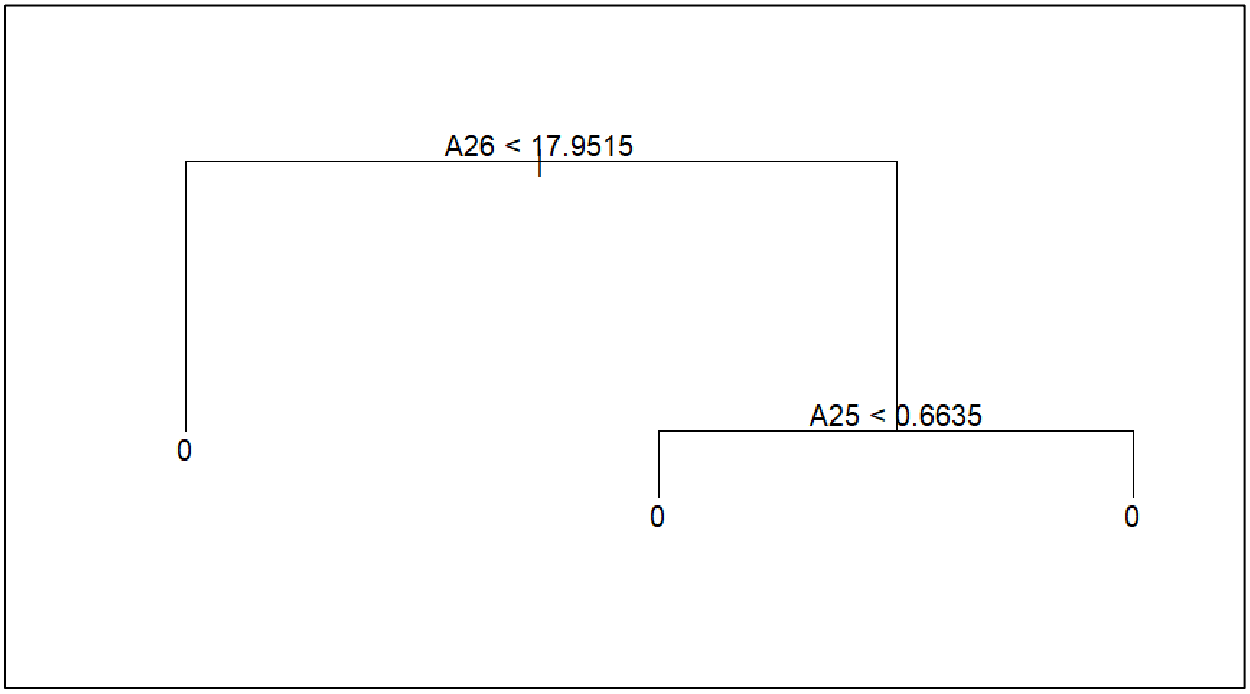
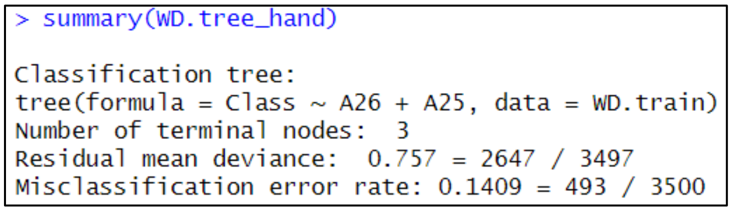


Figure 10: Simple Decision Tree Created Using A26, A25, and A02

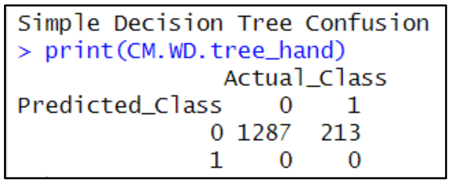
As shown in Figure 11, the attributes used to create this simple decision tree were A26 and A25. This aligns with the findings from Question 8, where A26 and A25 consistently appeared as top predictors across all classifiers. While A12 was also highlighted in the decision tree's importance ranking, it was not selected here due to its lower consistency across other models. Overall, the tree prioritized variables that provided the most predictive power while keeping the structure simple and interpretable.



```
> summary(WD.tree_hand)

Classification tree:
tree(formula = Class ~ A26 + A25, data = WD.train)
Number of terminal nodes:  3
Residual mean deviance:  0.757 = 2647 / 3497
Misclassification error rate: 0.1409 = 493 / 3500
```

Figure 11: Simple Decision Tree Most Important Attributes



```
Simple Decision Tree Confusion
> print(CM.WD.tree_hand)
               Actual_Class
Predicted_Class    0    1
              0 1287  213
              1    0    0
```

Figure 12: Confusion Matrix of Simple Decision Tree

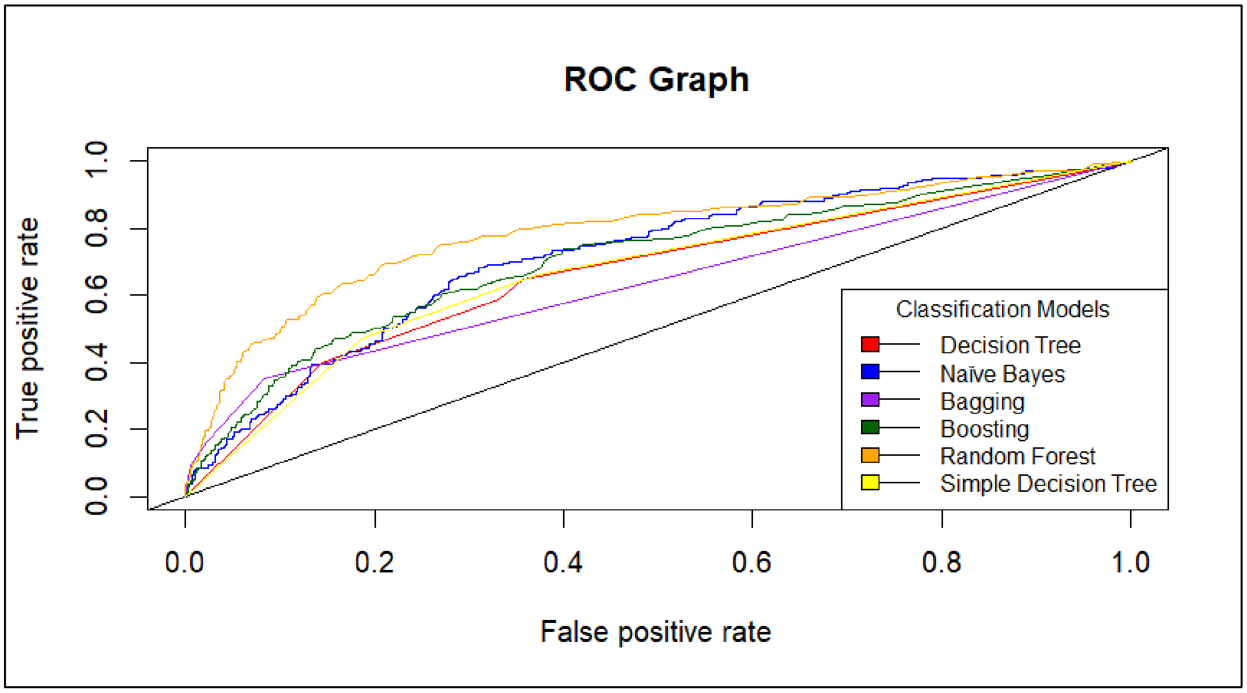
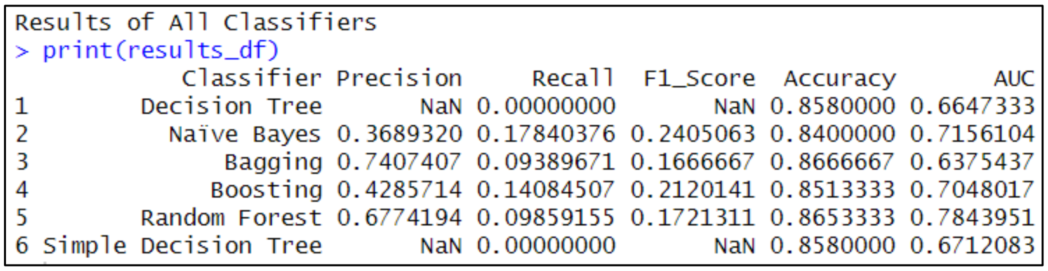


Figure 13: ROC Graph of All Classifiers including the Simple Decision Tree



```
Results of All Classifiers
> print(results_df)
            Classifier Precision     Recall  F1_Score  Accuracy       AUC
1        Decision Tree       NaN 0.00000000       NaN 0.8580000 0.6647333
2          Naïve Bayes 0.3689320 0.17840376 0.2405063 0.8400000 0.7156104
3              Bagging 0.7407407 0.09389671 0.1666667 0.8666667 0.6375437
4             Boosting 0.4285714 0.14084507 0.2120141 0.8513333 0.7048017
5        Random Forest 0.6774194 0.09859155 0.1721311 0.8653333 0.7843951
6 Simple Decision Tree       NaN 0.00000000       NaN 0.8580000 0.6712083
```

Figure 14: Table of Results of All Classifiers from Question 4 and the Simple Decision Tree

As shown in Figure 12, the confusion matrix for the simple decision tree is identical to that of the decision tree from Question 4, with 1,287 true negatives and 213 false negatives as shown in Figure 3. Similarly, the ROC curve in Figure 13 appears nearly the same as that of the original tree. Figure 14 further supports this similarity, as the precision, recall, F1 score, and accuracy values are the same for both models. However, the AUC value of the simple decision tree is slightly higher, indicating a marginally better ability to distinguish between the two classes. When compared to other models, the simple decision tree performs better in terms of accuracy than both the Naïve Bayes and Boosting classifiers. Additionally, it achieves a higher AUC than the Bagging model. This suggests that despite its simplicity, the model performs competitively and offers a good balance between interpretability and predictive performance.

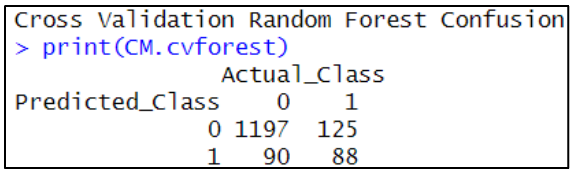
Question 11:

To create my improved model, I first addressed the issue of class imbalance in the training data. Specifically, I performed data balancing by oversampling more rows of the minority class, “Oats” (class = 1), into the training set. This helped ensure that the model had enough examples of both classes during training, improving its ability to learn meaningful patterns for “Oats” and reducing bias toward the majority class.

In addition to balancing the data, I implemented K-fold Cross Validation using the Random Forest algorithm, following the method described by Johnson (2024). This approach was used to improve reliability and generalizability. Unlike a single train-test split, K-Fold Cross-Validation trains and validates the model on multiple subsets of the data, providing a more robust estimate of performance. Since the class imbalance was addressed prior to training, this approach further ensures that the model's evaluation is not biased by a particular data split and performs consistently across different segments of the data. Random Forest was chosen for its strong performance as discussed in Question 7, while the cross-validation process reduces overfitting and increases confidence in the model's predictive ability on unseen data.

One of the main factors that influenced my decision was the poor performance of the models in Question 4 when predicting the “Other” class (class = 0). Across all classifiers, the recall and F1 scores were consistently low, indicating that the models struggled to correctly identify instances of the “Other” class. This imbalance in performance suggested that the models were biased toward predicting “Oats” (class = 1), likely due to class imbalance in the training data. Even when accuracy appeared high, it did not reflect the model’s true ability to distinguish between both classes effectively. Therefore, it was important to address this issue by both balancing the training data and choosing an evaluation strategy (K-Fold Cross-Validation) and model (Random Forest) that could better handle complex patterns while aiming for more balanced performance. My goal was to improve not just overall accuracy, but also the model’s ability to fairly and reliably predict both classes, which is reflected in improved recall and F1 score for the minority class.

The attributes used in the improved model were selected based on the variable importance analysis conducted in Question 8. I chose to include all attributes except those identified as insignificant across multiple classifiers—namely, A23, A20, A10, A28, and A15. These attributes consistently showed very low importance values in decision trees, bagging, boosting, and random forest models, and were not influential in predicting either class (“Oats” or “Other”). Excluding them helped reduce noise and potential overfitting while simplifying the model. On the other hand, attributes such as A26, A25, and A02 were retained because they were consistently ranked as the most important across most classifiers, indicating their strong predictive power. By focusing on these key variables, I aimed to improve model performance and interpretability without sacrificing accuracy or generalization.



```
Cross Validation Random Forest Confusion
> print(CM.cvforest)
               Actual_Class
Predicted_Class    0    1
              0 1197  125
              1   90   88
```

Figure 15: Confusion Matrix of K-fold Cross Validation Random Forest (k = 2)

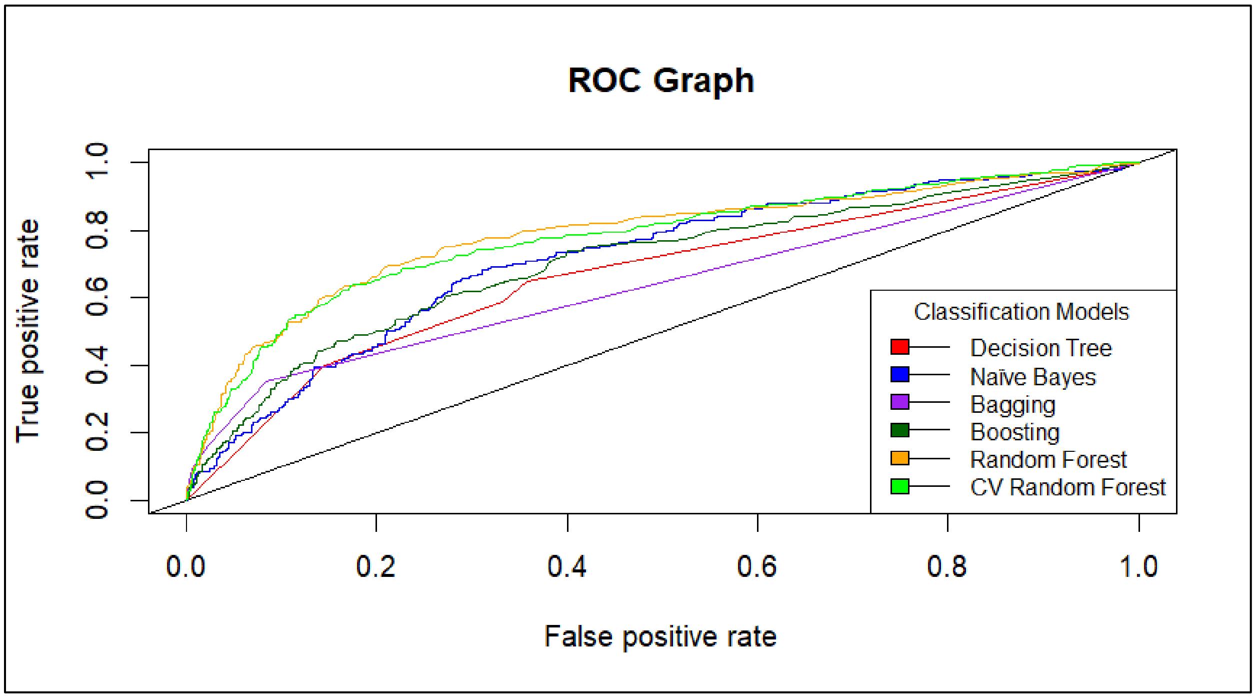


Figure 16: ROC Graph of All Classification Models including Cross Validation Random Forest

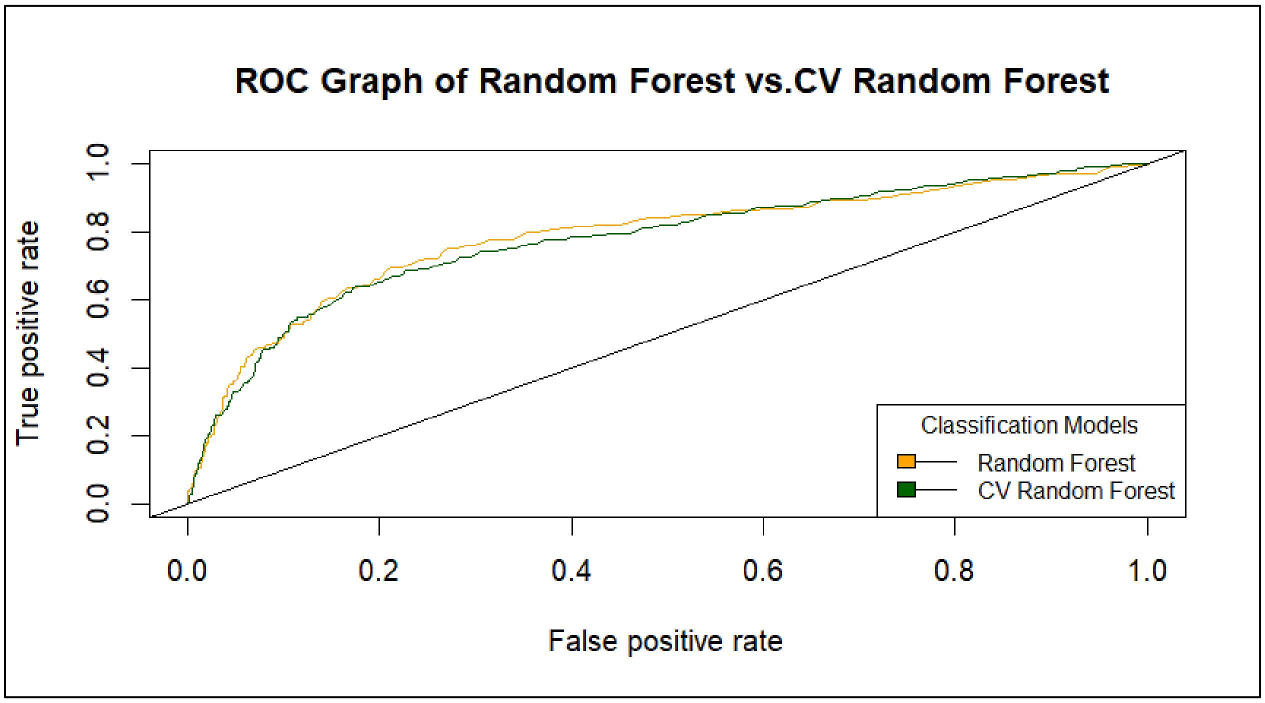
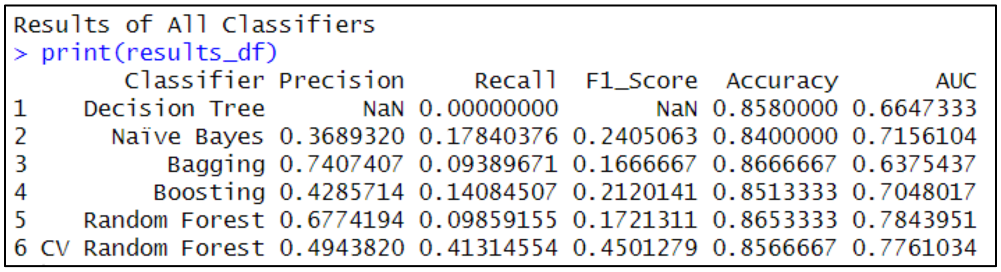


Figure 17: ROC Graph of Random Forest vs. Cross Validation Random Forest



```
Results of All Classifiers
> print(results_df)
        Classifier Precision     Recall  F1_Score  Accuracy       AUC
1    Decision Tree       NaN 0.00000000       NaN 0.8580000 0.6647333
2      Naïve Bayes 0.3689320 0.17840376 0.2405063 0.8400000 0.7156104
3          Bagging 0.7407407 0.09389671 0.1666667 0.8666667 0.6375437
4         Boosting 0.4285714 0.14084507 0.2120141 0.8513333 0.7048017
5    Random Forest 0.6774194 0.09859155 0.1721311 0.8653333 0.7843951
6 CV Random Forest 0.4943820 0.41314554 0.4501279 0.8566667 0.7761034
```

Figure 18: Results Table of All Classifiers including Cross Validation Random Forest

From Figures 16 and 17, it is observed that the ROC curves for the Cross-Validation Random Forest and the original Random Forest are very similar, suggesting both models have comparable abilities in distinguishing between the "Oats" and "Other" classes across different threshold values. However, the performance table shown in Figure 18 highlights a notable trade-off between accuracy and the balance between precision and recall. As the values of precision and recall become more aligned in the Cross-Validated Random Forest, the F1 score increases. This balance is critical, especially in cases where both false positives and false negatives are important to minimize.

This improvement is evident when comparing the F1 scores of the original Random Forest and the Cross-Validated version. While the original model has higher precision and accuracy, its recall was relatively poor, resulting in a lower F1 score. On the other hand, the Cross-Validated model achieves a more balanced performance, making it more effective at consistently identifying both "Oats" and "Other" classes.

Although the Cross-Validated Random Forest shows a slight decrease in overall accuracy and AUC compared to the original model, the gain in F1 score and the improved balance between class predictions are significant. This makes it a stronger and more reliable model, particularly in real-world applications where class imbalance is a concern. Therefore, I consider the Cross-Validation Random Forest the best-performing model overall, as it addresses the key weaknesses of the previous classifiers by improving generalization and fairness in prediction.

Question 12:

As mentioned in Question 11, the model excludes the insignificant attributes identified in Question 8 to improve performance and reduce complexity. By focusing on the most relevant features, the model is better able to capture meaningful patterns in the data without being distracted by noise from less important variables.

Several key pre-processing steps were applied to prepare the data for modeling. First, data balancing was performed by oversampling the minority class, "Oats", to address the class imbalance issue. This ensured that both classes were fairly represented during training, which is important for improving the model's ability to correctly detect the minority class and enhance metrics like recall and F1-score. Additionally, the Class variable was converted to a numeric format. This step was necessary because some modeling functions used in the "neuralnet" package, require numeric inputs for classification labels. Converting the class labels to numeric helped prevent errors and allowed for consistent prediction and evaluation across models.

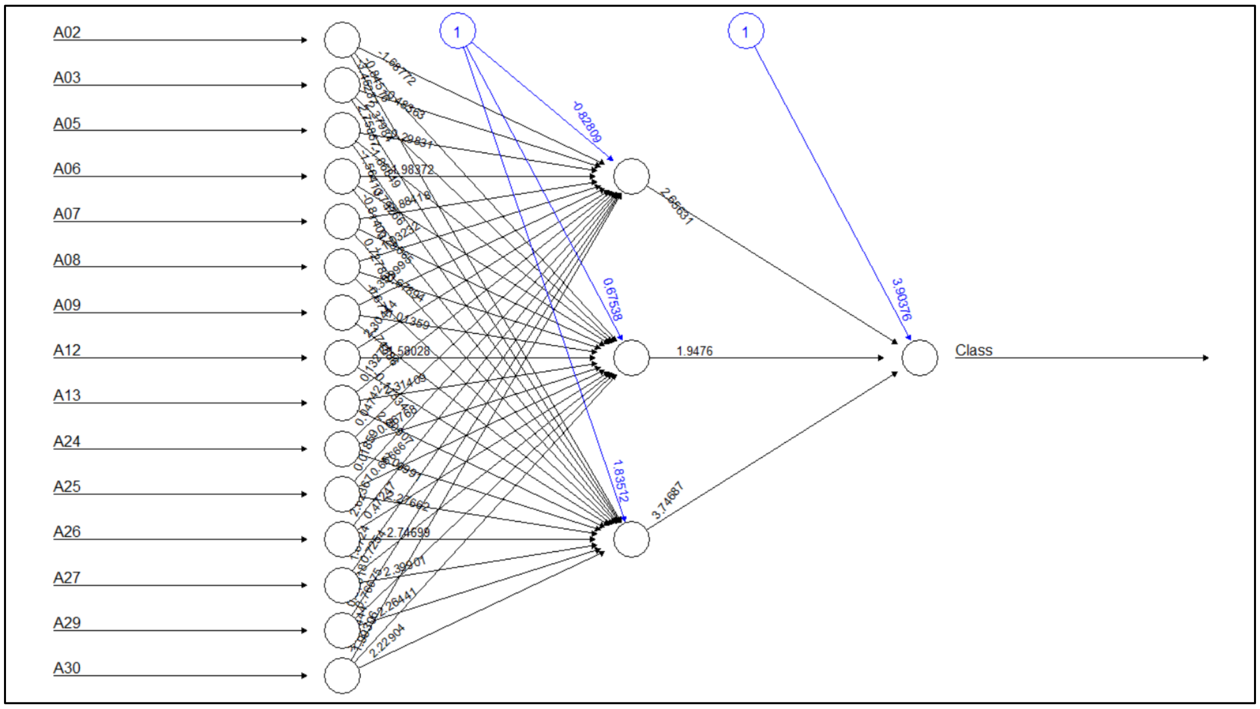


Figure 19: Artificial Neural Network Graph

Figure 19 shows the Artificial Neural Network graph plotted. The graph has 3 hidden nodes and 1 output node.

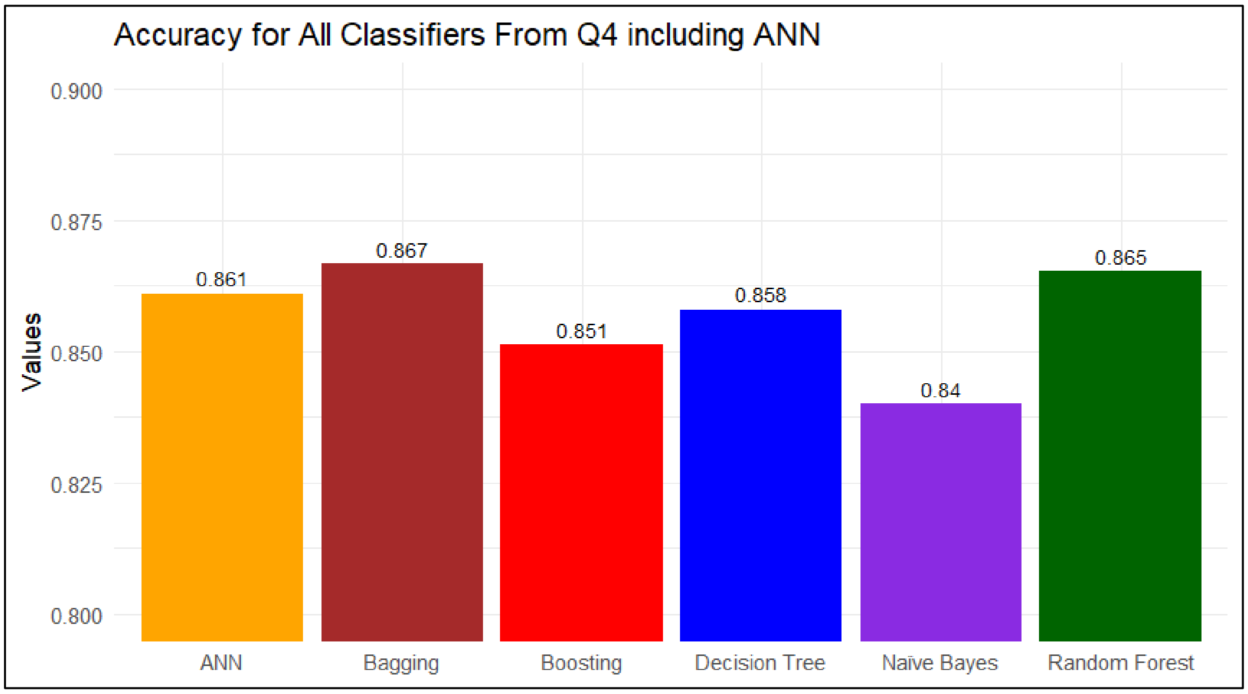


Figure 20: Bar Chart of Accuracy Values for All Classifiers including ANN

Based on Figure 20, the Artificial Neural Network (ANN) achieves quite high accuracy, ranking as the third highest. This value in accuracy could be due to several reasons. The model is capable of capturing complex, nonlinear relationships in the data which simpler models might miss. It can learn intricate patterns through its layered structure and nonlinear activation functions. Additionally, the data pre-processing steps such as balancing the classes may have helped the ANN model to perform better by providing a more representative training set. Moreover, its methods are robust to noise and irrelevant features, especially after excluding insignificant attributes identified in earlier analyses. These factors combined likely contribute to the comparable and strong accuracy performance observed in the ANN model.

Question 13:

The new classifier used in this implementation is the Support Vector Machine (SVM). To build the model, I used the "kernlab" package in R. This package provides functionality for kernel-based machine learning methods, including support vector machines. More information on the package and its capabilities can be found at the following link:

https://www.rdocumentation.org/packages/kernlab/versions/0.6-1/topics/ksvm

Support Vector Machines work by finding the optimal hyperplane that separates different classes in a dataset. As explained by Navlani (2019), the key idea behind SVM is to identify a decision boundary (hyperplane) that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class, known as support vectors. The process typically follows two main steps. First, the algorithm generates multiple possible hyperplanes that could separate the classes. For example, among several candidate lines (black, blue, and orange), only one may properly classify the training data. Second, the algorithm selects the hyperplane that offers the maximum margin from the closest data points of each class, as this is expected to generalize best to unseen data.

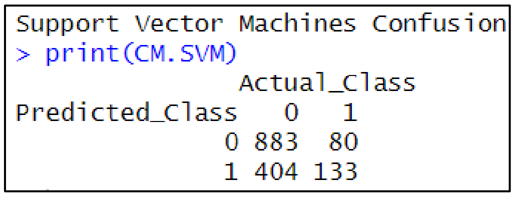
This characteristic of SVM makes it particularly powerful in cases where a clear margin of separation exists between classes, and it is capable of handling high-dimensional data using kernel functions.

The data processing steps taken to create this model began with converting the Class attribute back to a factor. This was necessary because the SVM classifier in R requires the response variable to be a factor when performing classification, as it needs to recognize the distinct classes ("0" for "Other" and "1" for "Oats") as categorical labels rather than continuous numeric values. Treating them as factors ensures that the model interprets the task correctly as a classification problem rather than regression.

Secondly, the model was trained using a dataset that had already been balanced. As previously discussed in Question 11, balancing the dataset during training is essential due to the original class imbalance, where the "Other" class (0) had significantly more instances than the "Oats" class (1).

Training on such imbalanced data would likely bias the model toward predicting the majority class, thereby reducing its ability to accurately classify the minority class. By using a balanced dataset, both classes are equally represented, enabling the model to learn from both categories more effectively and improving its ability to generalize to unseen data.

Lastly, I omitted the attributes A23, A20, A10, A28, and A15 when training the model. This decision was based on the results from Question 8, where these attributes were identified as having low importance to the model's performance. Removing these insignificant features helps simplify the model, reduce overfitting, and improve training efficiency without sacrificing predictive accuracy.



```
Support Vector Machines Confusion
> print(CM.SVM)
               Actual_Class
Predicted_Class   0   1
              0 883  80
              1 404 133
```

Figure 21: Confusion Matrix of Support Vector Machine

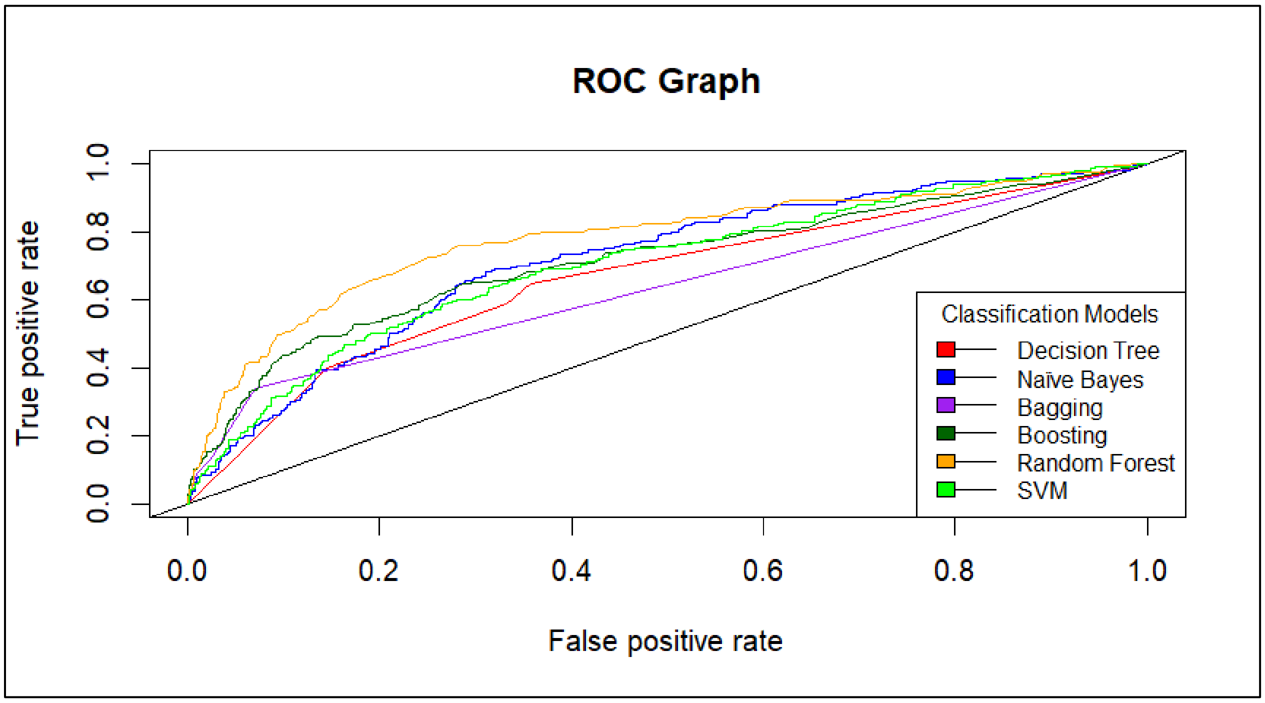


Figure 22: ROC Graph of All Classfiers including Support Vector Machine

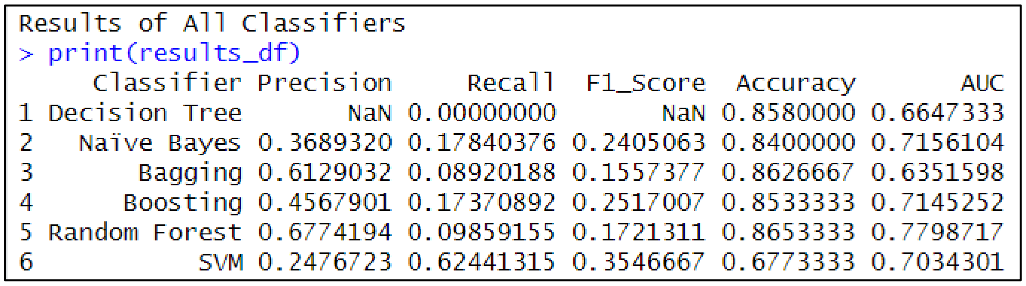

```
Results of All Classifiers
> print(results_df)
     Classifier Precision     Recall  F1_Score  Accuracy       AUC
1 Decision Tree       NaN 0.00000000       NaN 0.8580000 0.6647333
2   Naïve Bayes 0.3689320 0.17840376 0.2405063 0.8400000 0.7156104
3       Bagging 0.6129032 0.08920188 0.1557377 0.8626667 0.6351598
4      Boosting 0.4567901 0.17370892 0.2517007 0.8533333 0.7145252
5 Random Forest 0.6774194 0.09859155 0.1721311 0.8653333 0.7798717
6           SVM 0.2476723 0.62441315 0.3546667 0.6773333 0.7034301
```

Figure 23: Results of All Classifiers including Support Vector Machine

Based on the Confusion Matrix shown in Figure 21, the SVM model displays a low number of true negatives and false negatives, while exhibiting a high number of false positives and a low number of true positives, especially when compared to the Confusion Matrices in Figure 5. This suggests that the SVM model struggles to accurately predict the "Oats" class (class = 1), often misclassifying them as "Other" (class = 0), while still performing relatively better at identifying the majority class. In other words, the model has difficulty distinguishing the minority class, leading to poor classification performance for "Oats."

Moreover, the ROC curve in Figure 22 shows that SVM performs similarly to the Naïve Bayes and Boosting models in terms of its ability to distinguish between the two classes across various thresholds. This is further supported by Figure 23, where the SVM exhibits a low precision but a relatively high recall. The high recall indicates that the SVM is better at identifying most of the actual "Oats" samples, but the low precision suggests that many of these predictions are incorrect. This trade-off shows that the SVM balances recall slightly better than most models, prioritizing the correct identification of the minority class even at the expense of making more false positive errors.

However, its overall accuracy is the lowest among all classifiers. This low accuracy likely reflects the high number of misclassifications, particularly due to false positives, which inflates the error rate. Despite this, the AUC value of the SVM is not the lowest. AUC evaluates the model's ranking performance rather than its specific thresholder predictions, so a reasonable AUC indicates that the SVM is still relatively effective at separating the classes when not constrained to a fixed threshold.

Therefore, while the SVM does not perform best in terms of accuracy or precision, its decent AUC and high recall suggest it may still be useful in scenarios where identifying the minority class is particularly important, and where the cost of false positives is acceptable. This could justify its use in a context where the goal is to reduce the number of false negatives for "Oats," despite a general trade-off in overall accuracy.

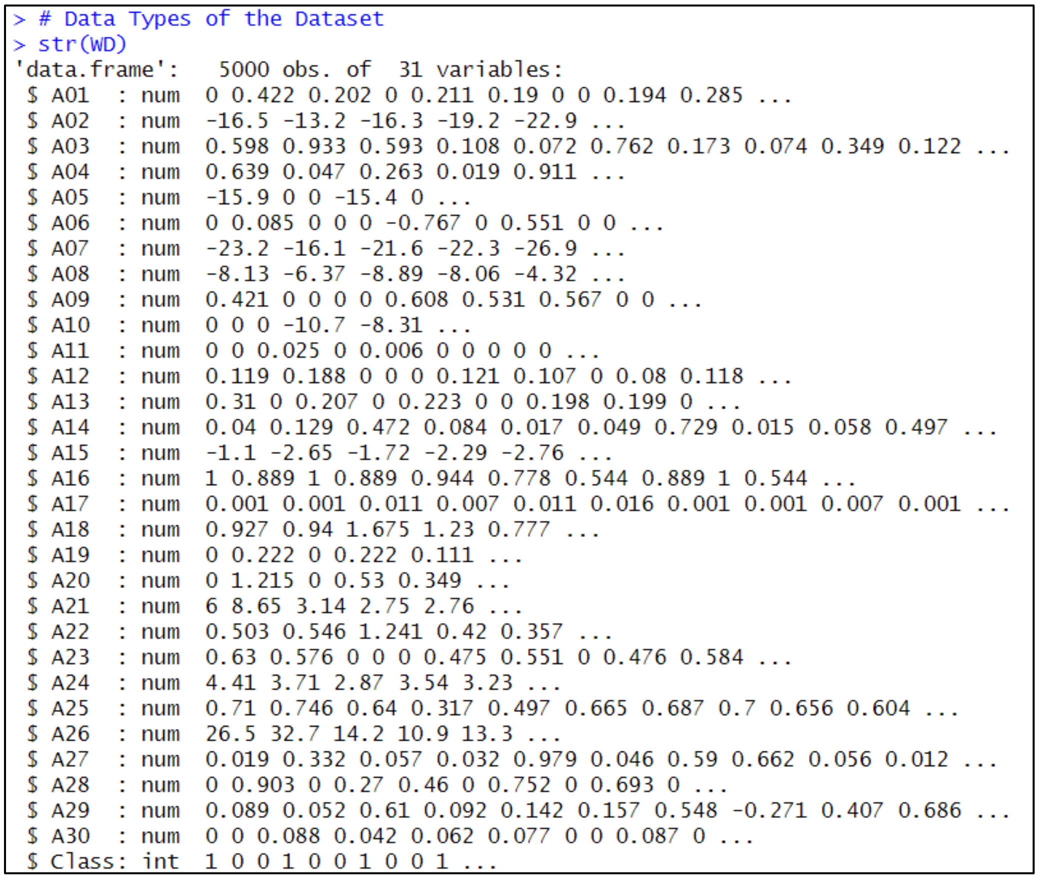
References

Johnson, D. (2024, June 12). *R random forest tutorial with example*. Guru99. https://www.guru99.com/r-random-forest-tutorial.html

Navlani, A. (2019, December 27). *Support Vector Machines with Scikit-learn Tutorial*. Datacamp. https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python#how-does-svm-work?-thema

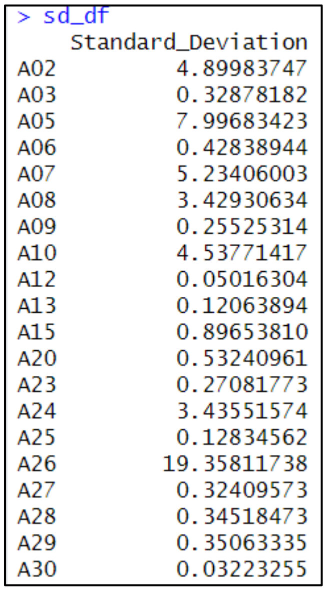
Appendix

Question 1's Dataset's Data Types Table



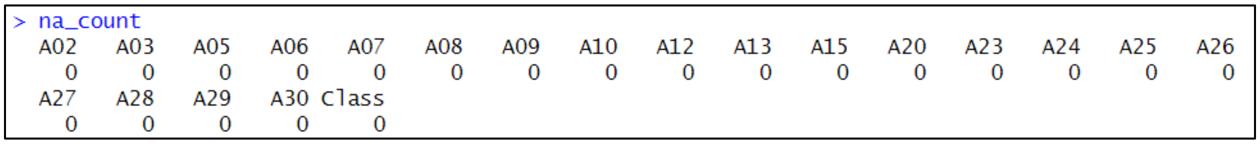
```
> # Data Types of the Dataset
> str(WD)
'data.frame':	5000 obs. of  31 variables:
 $ A01  : num  0 0.422 0.202 0 0.211 0.19 0 0 0.194 0.285 ...
 $ A02  : num  -16.5 -13.2 -16.3 -19.2 -22.9 ...
 $ A03  : num  0.598 0.933 0.593 0.108 0.072 0.762 0.173 0.074 0.349 0.122 ...
 $ A04  : num  0.639 0.047 0.263 0.019 0.911 ...
 $ A05  : num  -15.9 0 0 -15.4 0 ...
 $ A06  : num  0 0.085 0 0 0 -0.767 0 0.551 0 0 ...
 $ A07  : num  -23.2 -16.1 -21.6 -22.3 -26.9 ...
 $ A08  : num  -8.13 -6.37 -8.89 -8.06 -4.32 ...
 $ A09  : num  0.421 0 0 0 0 0.608 0.531 0.567 0 0 ...
 $ A10  : num  0 0 0 -10.7 -8.31 ...
 $ A11  : num  0 0 0.025 0 0.006 0 0 0 0 0 ...
 $ A12  : num  0.119 0.188 0 0 0 0.121 0.107 0 0.08 0.118 ...
 $ A13  : num  0.31 0 0.207 0 0.223 0 0 0.198 0.199 0 ...
 $ A14  : num  0.04 0.129 0.472 0.084 0.017 0.049 0.729 0.015 0.058 0.497 ...
 $ A15  : num  -1.1 -2.65 -1.72 -2.29 -2.76 ...
 $ A16  : num  1 0.889 1 0.889 0.944 0.778 0.544 0.889 1 0.544 ...
 $ A17  : num  0.001 0.001 0.011 0.007 0.011 0.016 0.001 0.001 0.007 0.001 ...
 $ A18  : num  0.927 0.94 1.675 1.23 0.777 ...
 $ A19  : num  0 0.222 0 0.222 0.111 ...
 $ A20  : num  0 1.215 0 0.53 0.349 ...
 $ A21  : num  6 8.65 3.14 2.75 2.76 ...
 $ A22  : num  0.503 0.546 1.241 0.42 0.357 ...
 $ A23  : num  0.63 0.576 0 0 0 0.475 0.551 0 0.476 0.584 ...
 $ A24  : num  4.41 3.71 2.87 3.54 3.23 ...
 $ A25  : num  0.71 0.746 0.64 0.317 0.497 0.665 0.687 0.7 0.656 0.604 ...
 $ A26  : num  26.5 32.7 14.2 10.9 13.3 ...
 $ A27  : num  0.019 0.332 0.057 0.032 0.979 0.046 0.59 0.662 0.056 0.012 ...
 $ A28  : num  0 0.903 0 0.27 0.46 0 0.752 0 0.693 0 ...
 $ A29  : num  0.089 0.052 0.61 0.092 0.142 0.157 0.548 -0.271 0.407 0.686 ...
 $ A30  : num  0 0 0.088 0.042 0.062 0.077 0 0 0.087 0 ...
 $ Class: int  1 0 0 1 0 0 1 0 0 1 ...
```

Question 1's Standard Deviation of Numerical Attributes



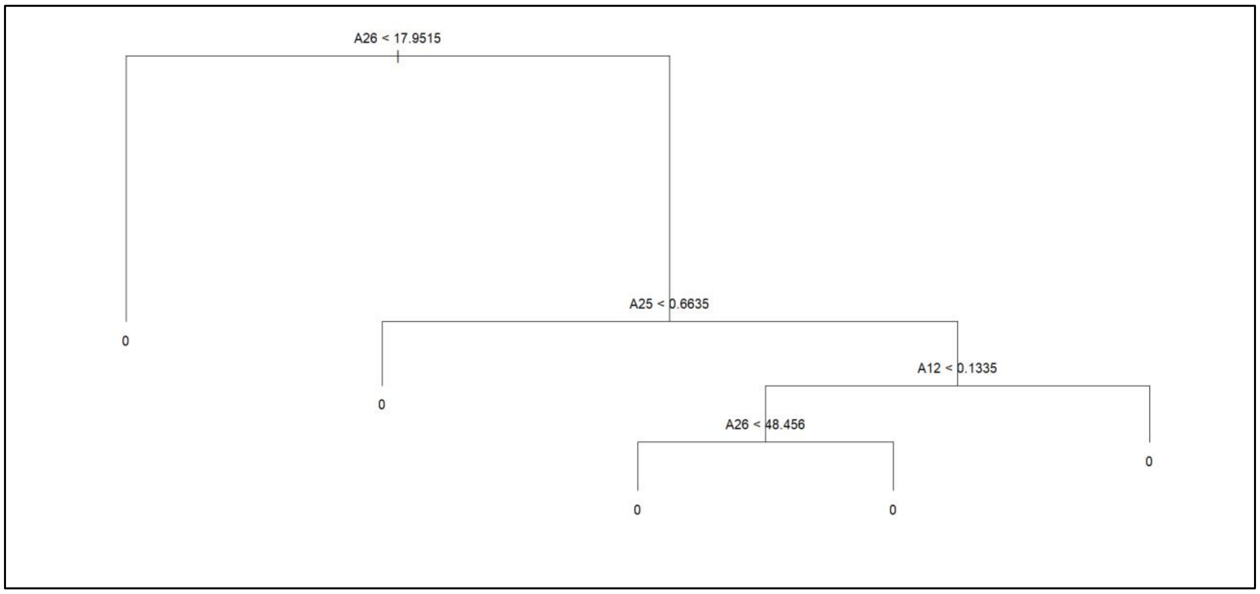
```
> sd_df
     Standard_Deviation
A02          4.89983747
A03          0.32878182
A05          7.99683423
A06          0.42838944
A07          5.23406003
A08          3.42930634
A09          0.25525314
A10          4.53771417
A12          0.05016304
A13          0.12063894
A15          0.89653810
A20          0.53240961
A23          0.27081773
A24          3.43551574
A25          0.12834562
A26         19.35811738
A27          0.32409573
A28          0.34518473
A29          0.35063335
A30          0.03223255
```

Question 2’s Missing Value Column Count



```
> na_count
  A02   A03   A05   A06   A07   A08   A09   A10   A12   A13   A15   A20   A23   A24   A25   A26 
    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0 
  A27   A28   A29   A30 Class 
    0     0     0     0     0 
```

Question 4’s Decision Tree



Question 1 R-code

```
rm(list = ls())
```

```
set.seed(33214476) # Your Student ID is the random seed
WD = read.csv("WinnData.csv")
WD = WD[sample(nrow(WD), 5000, replace = FALSE),]
WD = WD[,c(sort(sample(1:30,20, replace = FALSE)), 31)]


# Question 1
# Data Types of the Dataset
str(WD)

# Proportion Table
prop_tab <- table(WD$Class) / nrow(WD)
prop_tab

# Bar Chart of Proportion Table
barplot(prop_tab, main = "Oats vs. Other",
        xlab = "Type of Crop", ylab = "Proportion",
        col = c("lightblue", "pink"),
        legend.text = TRUE,
        ylim = c(0, 1.0))

# Distribution of Dataset
summary(WD[, 1:20])

# Standard Deviation of Numerical Attributes in Dataset
sd_vals <- sapply(WD, sd, na.rm = TRUE)

sd_df <- data.frame(Standard_Deviation = sd_vals)
sd_df
```

Question 2 R-code

```
# Counting missing values in each column
na_count <- colSums(is.na(WD))
na_count

# Factorising "Class" column
WD$Class <- as.factor(WD$Class)
str(WD$Class)
```

Question 3 R-code

```
set.seed(33214476)
train.row = sample(1:nrow(WD), 0.7*nrow(WD))
WD.train = WD[train.row,]
WD.test = WD[-train.row,]
```

Question 4 R-code

```
library(tree)
library(e1071)
library(ROCR)
library(randomForest)
library(adabag)
library(rpart)

# Decision Tree
WD.tree = tree(Class ~., data=WD.train)
WD.tree
summary(WD.tree)
plot(WD.tree)
text(WD.tree, pretty=0)

# Naïve Bayes
WD.bayes = naiveBayes(Class ~ ., data=WD.train)
WD.bayes
summary(WD.bayes)

# Bagging
WD.bag = bagging(Class ~ ., data=WD.train, mfinal=5)
WD.bag
summary(WD.bag)

# Boosting
WD.boost = boosting(Class ~ ., data=WD.train, mfinal=10)
WD.boost
summary(WD.boost)

# Random Forest
WD.forest =  randomForest(Class ~ ., data=WD.train)
WD.forest
summary(WD.forest)
```

Question 5 R-code

```
# Decision Tree:
# Confusion matrix
WD.predtree = predict(WD.tree, WD.test, type = "class")
CM.WD.tree = table(Predicted_Class = WD.predtree, Actual_Class
= WD.test$Class)
cat("\nDecision Tree Confusion\n")
print(CM.WD.tree)
```

```
# Precision
tree.prec = CM.WD.tree[4]/sum(CM.WD.tree[2], CM.WD.tree[4])
tree.prec

# Recall
tree.rec = CM.WD.tree[4]/sum(CM.WD.tree[3], CM.WD.tree[4])
tree.rec

# Accuracy
tree.acc = (sum(diag(CM.WD.tree))/sum(CM.WD.tree))
tree.acc

# F1-score
tree.f1 = (2 * tree.prec * tree.rec) / sum(tree.prec, tree.rec)
tree.f1

# Naïve Bayes:
# Confusion Matrix
WD.predbayes = predict(WD.bayes, WD.test)
CM.WD.bayes = table(Predicted_Class = WD.predbayes,
Actual_Class = WD.test$Class)
cat("\nNaïve Bayes Confusion\n")
print(CM.WD.bayes)

# Precision
bayes.prec = CM.WD.bayes[4]/sum(CM.WD.bayes[2], CM.WD.bayes[4])
bayes.prec

# Recall
bayes.rec = CM.WD.bayes[4]/sum(CM.WD.bayes[3], CM.WD.bayes[4])
bayes.rec

# Accuracy
bayes.acc = (sum(diag(CM.WD.bayes))/sum(CM.WD.bayes))
bayes.acc

# F1-score
bayes.f1 = (2 * bayes.prec * bayes.rec) / sum(bayes.prec,
bayes.rec)
bayes.f1

# Bagging:
# Confusion Matrix
WD.predbag = predict.bagging(WD.bag, WD.test)
CM.WD.bag = WD.predbag$confusion
cat("\nBagging Confusion\n")
```

```
print(CM.WD.bag)

# Precision
bag.prec = CM.WD.bag[4]/sum(CM.WD.bag[2], CM.WD.bag[4])
bag.prec

# Recall
bag.rec = CM.WD.bag[4]/sum(CM.WD.bag[3], CM.WD.bag[4])
bag.rec

# Accuracy
bag.acc = (sum(diag(CM.WD.bag))/sum(CM.WD.bag))
bag.acc

# F1-score
bag.f1 = (2 * bag.prec * bag.rec) / sum(bag.prec, bag.rec)
bag.f1

#Boosting:
# Confusion Matrix
WD.predboost = predict.boosting(WD.boost, newdata=WD.test)
CM.WD.boost = WD.predboost$confusion
cat("\nBoosting Confusion\n")
print(CM.WD.boost)

# Precision
boost.prec = CM.WD.boost[4]/sum(CM.WD.boost[2], CM.WD.boost[4])
boost.prec

# Recall
boost.rec = CM.WD.boost[4]/sum(CM.WD.boost[3], CM.WD.boost[4])
boost.rec

# Accuracy
boost.acc = (sum(diag(CM.WD.boost))/sum(CM.WD.boost))
boost.acc

# F1-score
boost.f1 = (2 * boost.prec * boost.rec) / sum(boost.prec,
boost.rec)
boost.f1

# Random Forest:
# Confusion Matrix
WD.predforest = predict(WD.forest, WD.test)
CM.WD.forest = table(Predicted_Class = WD.predforest,
Actual_Class = WD.test$Class)
```

```
CM.WD.forest

# Precision
forest.prec = CM.WD.forest[4]/sum(CM.WD.forest[2],
CM.WD.forest[4])
forest.prec

# Recall
forest.rec = CM.WD.forest[4]/sum(CM.WD.forest[3],
CM.WD.forest[4])
forest.rec

# Accuracy
forest.acc = (sum(diag(CM.WD.forest))/sum(CM.WD.forest))
forest.acc

# F1-score
forest.f1 = (2 * forest.prec * forest.rec) / sum(forest.prec,
forest.rec)
forest.f1
```

Question 6 R-code

```
# Decision Tree:
# ROC Graph
WD.predtree.vector = predict(WD.tree, WD.test, type="vector")
tree.pred = prediction(WD.predtree.vector[,2], WD.test$Class)
tree.perf = performance(tree.pred, "tpr", "fpr")
plot(tree.perf, col="red", main = "ROC Graph")
abline(0,1)

# AUC Calculation
tree.auc = performance(tree.pred, "auc")
tree.auc_value = as.numeric(tree.auc@y.values[[1]])
tree.auc_value

# Naïve Bayes:
# ROC Graph
WD.predbayes.raw = predict(WD.bayes, WD.test, type = "raw")
bayes.pred = prediction(WD.predbayes.raw[,2], WD.test$Class)
bayes.perf = performance(bayes.pred, "tpr", "fpr")
plot(bayes.perf, add=TRUE, col = "blue")

# AUC Calculation
bayes.auc = performance(bayes.pred, "auc")
bayes.auc_value = as.numeric(bayes.auc@y.values[[1]])
bayes.auc_value
```

```
# Bagging:
# ROC Graph
bag.pred = prediction(WD.predbag$prob[,2], WD.test$Class)
bag.perf = performance(bag.pred, "tpr", "fpr")
plot(bag.perf, add=TRUE, col = "purple")

# AUC Calculation
bag.auc = performance(bag.pred, "auc")
bag.auc_value = as.numeric(bag.auc@y.values[[1]])
bag.auc_value

# Boosting:
# ROC Graph
boost.pred = prediction(WD.predboost$prob[,2], WD.test$Class)
boost.perf = performance(boost.pred, "tpr", "fpr")
plot(boost.perf, add=TRUE, col = "darkgreen")

# AUC Calculation
boost.auc = performance(boost.pred, "auc")
boost.auc_value = as.numeric(boost.auc@y.values[[1]])
boost.auc_value

# Random Forest:
# ROC Graph
WD.predforest.prob = predict(WD.forest, WD.test, type="prob")
forest.pred = prediction( WD.predforest.prob[,2],
WD.test$Class)
forest.perf = performance(forest.pred,"tpr","fpr")
plot(forest.perf, add=TRUE, col = "orange")

# AUC Calculations
forest.auc = performance(forest.pred, "auc")
forest.auc_value = as.numeric(forest.auc@y.values[[1]])
forest.auc_value

# Legend to overall Plot
legend(x = "bottomright", title="Classification Models",
       legend=c("Decision Tree","Naïve
Bayes","Bagging","Boosting","Random Forest"),
       fill=c("red",
"blue","purple","darkgreen","orange"),lty=1,cex=0.8)
```

Question 7 R-code

```
classifiers = c("Decision Tree", "Naïve Bayes",
"Bagging","Boosting","Random Forest")
```

```
precision_results = c(tree.prec, bayes.prec, bag.prec,
boost.prec, forest.prec)
recall_results = c(tree.rec, bayes.rec, bag.rec, boost.rec,
forest.rec)
f1_results = c(tree.f1, bayes.f1, bag.f1, boost.f1, forest.f1)
accuracy_results = c(tree.acc, bayes.acc, bag.acc, boost.acc,
forest.acc)
auc_results = c(tree.auc_value, bayes.auc_value, bag.auc_value,
boost.auc_value, forest.auc_value)

results_df = data.frame(Classifier = classifiers,
                        Precision = precision_results,
                        Recall = recall_results,
                        F1_Score = f1_results,
                        Accuracy = accuracy_results,
                        AUC = auc_results)
cat("\nResults of All Classifiers\n")
print(results_df)
```

Question 8 R-code

```
print(summary(WD.tree))
print(WD.bag$importance)
print(WD.boost$importance)
print(WD.forest$importance)
print(order(WD.forest$importance))
```

Question 10 R-code

```
# Question 10
WD.tree_hand = tree(Class ~A26+A25+A02, data=WD.train)
summary(WD.tree_hand)
plot(WD.tree_hand)
text(WD.tree_hand, pretty=0)

# Confusion matrix
WD.predtree_hand = predict(WD.tree_hand, WD.test, type =
"class")
CM.WD.tree_hand = table(Predicted_Class = WD.predtree_hand,
Actual_Class = WD.test$Class)
cat("\nSimple Decision Tree Confusion\n")
print(CM.WD.tree_hand)

# Precision
tree_hand.prec = CM.WD.tree_hand[4]/sum(CM.WD.tree_hand[2],
CM.WD.tree_hand[4])
```

```
tree_hand.prec

# Recall
tree_hand.rec = CM.WD.tree_hand[4]/sum(CM.WD.tree_hand[3],
CM.WD.tree_hand[4])
tree_hand.rec

# Accuracy
tree_hand.acc =
(sum(diag(CM.WD.tree_hand))/sum(CM.WD.tree_hand))
tree_hand.acc

# F1-score
tree_hand.f1 = (2 * tree_hand.prec * tree_hand.rec) /
sum(tree_hand.prec, tree_hand.rec)
tree_hand.f1

# ROC Graph
WD.predtree_hand.vector = predict(WD.tree_hand, WD.test,
type="vector")
tree_hand.pred = prediction(WD.predtree_hand.vector[,2],
WD.test$Class)
tree_hand.perf = performance(tree_hand.pred, "tpr", "fpr")
plot(tree.perf, col="red", main = "ROC Graph")
abline(0,1)
plot(bayes.perf, add=TRUE, col = "blue")
plot(bag.perf, add=TRUE, col = "purple")
plot(boost.perf, add=TRUE, col = "darkgreen")
plot(forest.perf, add=TRUE, col = "orange")
plot(tree_hand.perf, add = TRUE, col = "yellow")

# Legend to overall Plot
legend(x = "bottomright", title="Classification Models",
       legend=c("Decision Tree","Naïve
Bayes","Bagging","Boosting","Random Forest", "Simple Decision
Tree"),
       fill=c("red", "blue","purple","darkgreen","orange",
"yellow"),lty=1,cex=0.8)
# Legend to overall Plot
legend(x = "bottomright", title="Classification Models",
       legend=c("Decision Tree","Naïve
Bayes","Bagging","Boosting","Random Forest", "Simple Decision
Tree"),
       fill=c("red", "blue","purple","darkgreen","orange",
"yellow"),lty=1,cex=0.8)

# AUC Calculation
```

```
tree_hand.auc = performance(tree_hand.pred, "auc")
tree_hand.auc_value = as.numeric(tree_hand.auc@y.values[[1]])
tree_hand.auc_value

# Results Table
classifiers = c("Decision Tree", "Naïve Bayes",
"Bagging","Boosting", "Random Forest", "Simple Decision Tree")
precision_results = c(tree.prec, bayes.prec, bag.prec,
boost.prec, forest.prec, tree_hand.prec)
recall_results = c(tree.rec, bayes.rec, bag.rec, boost.rec,
forest.rec, tree_hand.rec)
f1_results = c(tree.f1, bayes.f1, bag.f1, boost.f1, forest.f1,
tree_hand.f1)
accuracy_results = c(tree.acc, bayes.acc, bag.acc, boost.acc,
forest.acc, tree_hand.acc)
auc_results = c(tree.auc_value, bayes.auc_value, bag.auc_value,
boost.auc_value, forest.auc_value, tree_hand.auc_value)

results_df = data.frame(Classifier = classifiers,
                        Precision = precision_results,
                        Recall = recall_results,
                        F1_Score = f1_results,
                        Accuracy = accuracy_results,
                        AUC = auc_results)
cat("\nResults of All Classifiers\n")
print(results_df)
```

Question 11 R-code

```
# Data Balancing
oats <- WD.train[WD.train$Class == 1, ]
other <- WD.train[WD.train$Class == 0, ]

set.seed(33214476)  # For reproducibility
oats_sampled  <- oats[sample(nrow(oats), 1750, replace = TRUE),
]
other_sampled <- other[sample(nrow(other), 1750, replace =
TRUE), ]

# Combine and shuffle rows
WD.train_balanced <- rbind(oats_sampled, other_sampled)
WD.train_balanced <-
WD.train_balanced[sample(nrow(WD.train_balanced)), ]

# Convert to data frame just in case
WD.train_balanced <- as.data.frame(WD.train_balanced)
```

```
cv_forest <- train(Class ~., data = WD.train_balanced, method =
"rf", metric = "Accuracy",
                    trControl = trainControl(method =
"cv",number = 10),
                    tuneLength = 10)
# Print the best parameters
cv_forest$bestTune

WD.cvrf =  randomForest(Class ~., data=WD.train_balanced, mtry
= 2)
WD.cvrf

#Make predictions using random forest
WD.predcvrf <- predict(WD.cvrf, WD.test)

#Confusion Matrix
CM.cvforest=table(Predicted_Class = WD.predcvrf, Actual_Class =
WD.test$Class)
cat("\nCross Validation Random Forest Confusion")
print(CM.cvforest)

# Precision
tree_best.prec = CM.cvforest[4]/sum(CM.cvforest[2],
CM.cvforest[4])
tree_best.prec

# Recall
tree_best.rec = CM.cvforest[4]/sum(CM.cvforest[3],
CM.cvforest[4])
tree_best.rec

# Accuracy
tree_best.acc = (sum(diag(CM.cvforest))/sum(CM.cvforest))
tree_best.acc

# F1-score
tree_best.f1 = (2 * tree_best.prec * tree_best.rec) /
sum(tree_best.prec, tree_best.rec)
tree_best.f1

#Plot ROC graph
WD.predcvrft.prob <- predict(WD.cvrf, WD.test, type="prob")
cvrf.pred <- prediction( WD.predcvrft.prob[,2], WD.test$Class)
cvrf.perf <- performance(cvrf.pred,"tpr","fpr")
plot(tree.perf, col="red", main = "ROC Graph")
abline(0,1)
plot(bayes.perf, add=TRUE, col = "blue")
plot(bag.perf, add=TRUE, col = "purple")
plot(boost.perf, add=TRUE, col = "darkgreen")
plot(forest.perf, add=TRUE, col = "orange")
plot(cvrf.perf, add=TRUE, col = "green")

# Legend to overall Plot
legend(x = "bottomright", title="Classification Models",
       legend=c("Decision Tree","Naïve
Bayes","Bagging","Boosting","Random Forest", "CV Random
Forest"),
       fill=c("red", "blue","purple","darkgreen","orange",
"green"),lty=1,cex=0.8)

# ROC Graph of Random Forest vs. CV Random Forest
plot(forest.perf, col = "orange", main = " ROC Graph of Random
Forest vs.CV Random Forest")
plot(cvrf.perf, add = TRUE, col = "darkgreen")
abline(0,1)

# Legend for plot above
legend(x = "bottomright", title="Classification Models",
       legend=c("Random Forest", "CV Random Forest"),
       fill=c("orange", "darkgreen"),lty=1,cex=0.8)

#Calculate AUC
tree_best.auc <- performance(cvrf.pred, "auc")
tree_best.auc_value <- as.numeric(tree_best.auc@y.values[[1]])
tree_best.auc_value

# Results Table
classifiers = c("Decision Tree", "Naïve Bayes",
"Bagging","Boosting", "Random Forest", "CV Random Forest")
precision_results = c(tree.prec, bayes.prec, bag.prec,
boost.prec, forest.prec, tree_best.prec)
recall_results = c(tree.rec, bayes.rec, bag.rec, boost.rec,
forest.rec, tree_best.rec)
f1_results = c(tree.f1, bayes.f1, bag.f1, boost.f1, forest.f1,
tree_best.f1)
accuracy_results = c(tree.acc, bayes.acc, bag.acc, boost.acc,
forest.acc, tree_best.acc)
auc_results = c(tree.auc_value, bayes.auc_value, bag.auc_value,
boost.auc_value, forest.auc_value, tree_best.auc_value)

results_df = data.frame(Classifier = classifiers,
                        Precision = precision_results,
                        Recall = recall_results,
                        F1_Score = f1_results,
                        Accuracy = accuracy_results,
                        AUC = auc_results)
cat("\nResults of All Classifiers\n")
print(results_df)
```

Question 12 R-code:

```
library(neuralnet)

# convert into numeric
WD$Class = as.numeric(WD$Class)
str(WD$Class)

#Training and testing rows
set.seed(33214476)
train.row_ANN = sample(1:nrow(WD), 0.8*nrow(WD))
WD.train_ANN = WD[train.row_ANN,]
WD.test_ANN = WD[-train.row_ANN,]

# Data Balancing
oats_ANN <- WD.train_ANN[WD.train_ANN$Class == 1, ]
other_ANN <- WD.train_ANN[WD.train_ANN$Class == 2, ]

set.seed(33214476)  # For reproducibility
oats_ANN_sampled  <- oats_ANN[sample(nrow(oats_ANN), 2000,
replace = TRUE), ]
other_ANN_sampled <- other_ANN[sample(nrow(other_ANN), 2000,
replace = TRUE), ]

# Combine and shuffle rows
WD.train_balanced_ANN <- rbind(oats_ANN_sampled,
other_ANN_sampled)
WD.train_balanced_ANN <-
WD.train_balanced_ANN[sample(nrow(WD.train_balanced_ANN)), ]

# Convert to data frame just in case
WD.train_balanced_ANN <- as.data.frame(WD.train_balanced_ANN)

#neural network
WD.nn = neuralnet(Class ~.-A23-A20-A10-A28-A15,
                  WD.train_balanced_ANN, hidden=3,linear.output
= FALSE)
WD.nn

#Plot NN
plot(WD.nn)
```

```
# Compute predictions
WD.nnpred = compute(WD.nn, WD.test_ANN)

# Round these to integers
WD.nnpredround = round(WD.nnpred$net.result, 0)

# Confusion Matrix
CM.nn = table(Predicted_Class = WD.nnpredround, Actual_Class =
WD.test_ANN$Class)
CM.nn

# Accuracy
nn.acc = (sum(diag(CM.nn))/sum(CM.nn))
nn.acc

# Accuracy results Bar Chart
q12_classifiers <- c("Decision Tree", "Naïve Bayes", "Bagging",
"Boosting", "Random Forest", "ANN")
q12_accuracy_results <- c(tree.acc, bayes.acc, bag.acc,
boost.acc, forest.acc, nn.acc)

q12_results_df <- data.frame(Classifier = q12_classifiers,
                            Accuracy = q12_accuracy_results,
                            Color = c("blue",
"blueviolet","brown","red","darkgreen","orange"))
q12_results_df

ggplot(data = q12_results_df, aes(x = Classifier, y = Accuracy,
fill = Color)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5, size
= 3) +
  labs(title = "Accuracy for All Classifiers From Q4 including
ANN",
       x = NULL,
       y = "Values",) +
  scale_fill_identity() +
  coord_cartesian(ylim=c(0.8,0.9)) +
  theme_minimal()
```

Question 13 R-code:

```
library(kernlab)

#Reassigning Class as factor
WD$Class <- as.factor(WD$Class)
```

```
WD.svm <- ksvm(Class ~ .-A23-A20-A10-A28-A15,
WD.train_balanced, kernel = "rbfdot", prob.model = TRUE)
WD.svm

# Make predictions using SVM
WD.predSVM = predict(WD.svm, WD.test)

# Confusion matrix
CM.SVM = table(Predicted_Class = WD.predSVM, Actual_Class =
WD.test$Class)
cat("\nSupport Vector Machines Confusion")
print(CM.SVM)

# Precision
SVM.prec = CM.SVM[4]/sum(CM.SVM[2], CM.SVM[4])
SVM.prec

# Recall
SVM.rec = CM.SVM[4]/sum(CM.SVM[3], CM.SVM[4])
SVM.rec

# Accuracy
SVM.acc = (sum(diag(CM.SVM))/sum(CM.SVM))
SVM.acc

# F1-score
SVM.f1 = (2 * SVM.prec * SVM.rec) / sum(SVM.prec, SVM.rec)
SVM.f1

#Plotting ROC graph
WD.SVM.prob = predict(WD.svm, WD.test, type = "probabilities")
pred.SVM <- prediction(WD.SVM.prob[,2], WD.test$Class)
perf.SVM <- performance(pred.SVM,"tpr","fpr")

plot(tree.perf, col="red", main = "ROC Graph")
abline(0,1)
plot(bayes.perf, add=TRUE, col = "blue")
plot(bag.perf, add=TRUE, col = "purple")
plot(boost.perf, add=TRUE, col = "darkgreen")
plot(forest.perf, add=TRUE, col = "orange")
plot(perf.SVM, add=TRUE, col = "green")

# Legend to overall Plot
legend(x = "bottomright", title="Classification Models",
       legend=c("Decision Tree","Naïve
Bayes","Bagging","Boosting","Random Forest", "SVM"),
       fill=c("red", "blue","purple","darkgreen","orange",
"green"),lty=1,cex=0.8)

# AUC
SVM.auc <- performance(pred.SVM, "auc")
SVM.auc_value <- as.numeric(SVM.auc@y.values[[1]])
SVM.auc_value

# Table of results
classifiers = c("Decision Tree", "Naïve Bayes",
"Bagging","Boosting", "Random Forest", "SVM")
precision_results = c(tree.prec, bayes.prec, bag.prec,
boost.prec, forest.prec, SVM.prec)
recall_results = c(tree.rec, bayes.rec, bag.rec, boost.rec,
forest.rec, SVM.rec)
f1_results = c(tree.f1, bayes.f1, bag.f1, boost.f1, forest.f1,
SVM.f1)
accuracy_results = c(tree.acc, bayes.acc, bag.acc, boost.acc,
forest.acc, SVM.acc)
auc_results = c(tree.auc_value, bayes.auc_value, bag.auc_value,
boost.auc_value, forest.auc_value, SVM.auc_value)

results_df = data.frame(Classifier = classifiers,
                        Precision = precision_results,
                        Recall = recall_results,
                        F1_Score = f1_results,
                        Accuracy = accuracy_results,
                        AUC = auc_results)
cat("\nResults of All Classifiers\n")
print(results_df)
```