Performance Evaluation of Supervised Learning Algorithms in Oats Classification

Mary Grace Paramitha Saraswati
(Student ID: 33410712)

Table of Contents

1. Data Exploration....2
2. Pre-Processing....3
3. Train Test Splitting....3
4. Classification Model Implementation....3
5. Classification Results....4
7. Which Model is the Best?....5
8. Attribute / Feature Importance....6
9. Justification on Model Performance Comparison....7
10. Simple Model....7
11. Best Tree-Based Classifier....8
12. Artificial Neural Network (ANN)....9
13. New Classifier (Support Vector Machine / SVM)....10
Appendix....12

1. Data Exploration

To begin the analysis, I explored the distribution of the target variable, Class, which indicates whether a particular crop is “Oats” (class = 1) or “Other” (class = 0). After sampling 5,000 rows from the original dataset, I found that the data was significantly imbalanced: only 13.9% of the entries were labeled as “Oats.” This imbalance raises concerns about model bias, as classifiers may be biased to favor the majority class (“Other”), while actually it is just due to “guessing”. The table below shows the difference in number of class=0 and class=1.

“Others”	**“Oats”**
86168	13832

Table 1.1: Distribution of “Oats” and “Others” class before balancing

Hence, corrective measures must be taken. The dataset was then balanced and the proportion of the “Oats” class increased to 50.9%. The table below shows the number of “Oats” and “Other” classes after sampling. See Appendix 1.1 for the bar plot and Pre-Processing section for the detailed steps.

“Others”	**“Oats”**
2455	2545

Table 1.2: Distribution of “Oats” and “Others” class after balancing and sampling

The dataset includes the following predictor (independent) variables: "A01", "A02", "A04", "A09", "A10", "A11", "A13", "A14", "A15", "A16", "A17", "A19", "A20", "A21", "A23", "A24", "A26", "A27", "A28", "A29". These variables are all continuous and are represented as floats. The dependent variable, “Class”, is a binary integer (0 or 1), representing whether a certain plant is an oat or not.

I also perform some data quality checks. There were no missing values found in the dataset. Variables A01, A09, A13, A17, and A23 seemed to display near-zero variance or low uniqueness relative to the sample size as shown in Appendix 1.2. These may contribute little to model performance, however this will not be omitted in initial analysis.

I used the boxplot method to detect outliers. Variables A01, A02, A11, A13, A15, A16, A17, A19, A21, and A29, exhibited outliers, and some (A11, A15, A16, A17, A19, A26) contained substantial outlier presence. Although these could affect model performance, this will not be omitted in initial analysis, and I treat them as part of the natural data variability or noise.

As seen in the correlation matrix in Figure 1.2, few variables showed significant correlation. These correlations suggest possible multicollinearity. However, this will be retained for initial analysis. Those variables are A16 and A19 (-0.974), and A16 and A20 (-0.908).

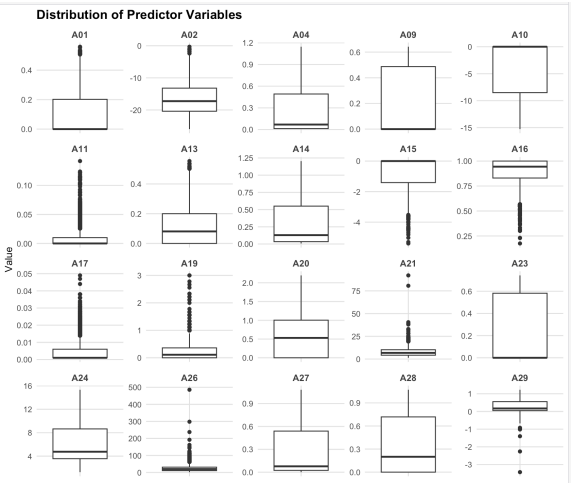


Figure 1.1: Distribution of predictors boxplot

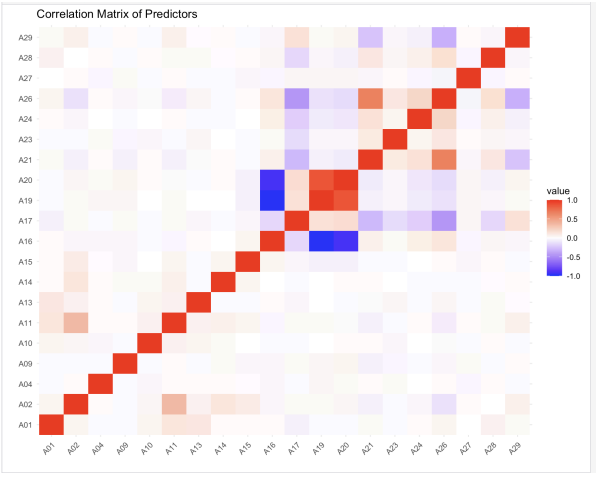


Figure 1.2: Correlation matrix of predictors

Next is the distribution analysis. The boxplots showed minimal differences in the distribution of variables between the two classes, except for A09, where a notable difference in medians was observed. This suggests A09 may serve as a strong predictor. Density plots indicated only minor distributional differences between classes for most variables. Slight separations were observed in A02, A23, A24, A26, and A28.

Although we retained the findings, these specific attributes must be taken note for further analysis and refinement.

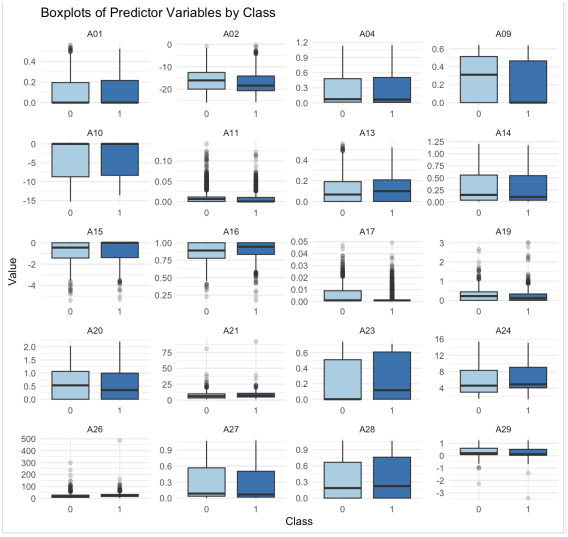


Figure 1.3: Distribution of predictors by class boxplot

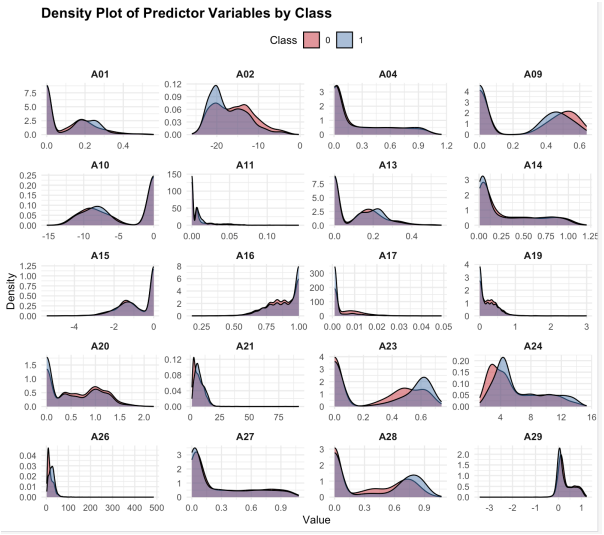


Figure 1.4: Density plot of predictors by class

2. Pre-Processing

Prior to model fitting, several preprocessing steps were carried out to prepare the dataset for analysis. Given the initial imbalanced data, the dataset was rebalanced to mitigate the risk of model bias toward the majority class, by duplicating existing rows belonging to the minority class, then shuffling the resulting dataset. After balancing, the proportion of the "Oats" class increased, creating a more equitable distribution across classes.

To ensure that all numeric variables contributed equally to the model and to improve convergence during training, the dataset was standardised. This transformation centered the data by subtracting the mean and scaled it to unit variance. Standardization is particularly important for models sensitive to feature magnitude, such as distance-based algorithms or those relying on gradient-based optimization.

3. Train Test Splitting

The dataset was divided into training and test subsets using a random sampling approach with a 70:30 ratio. A fixed random seed (my student ID) was used to ensure the results are reproducible. The training set consists of 3,500 rows and 21 variables. The test set consists of 1,500 rows and 21 variables, and is reserved for evaluating model performance on previously unseen data.

4. Classification Model Implementation

A variety of classification techniques were implemented to compare performance across different models. Each method was initially executed using default parameters.

a. **Decision Tree:**
   A classification tree was built using the tree function from the tree package in R. The model was initially trained with default settings. To improve performance and prevent overfitting, the tree was pruned using 10-fold cross-validation, which determined the

optimal number of terminal nodes. It is suggested that the optimal tree size is 4 terminal nodes, as shown in the graph in Appendix 2.1.4.

After creating the tree, it is shown that the key attributes used were A26, A11, and A09. This indicates that they had the highest discriminatory power in the dataset. A visual representation of the final tree structure is shown below.

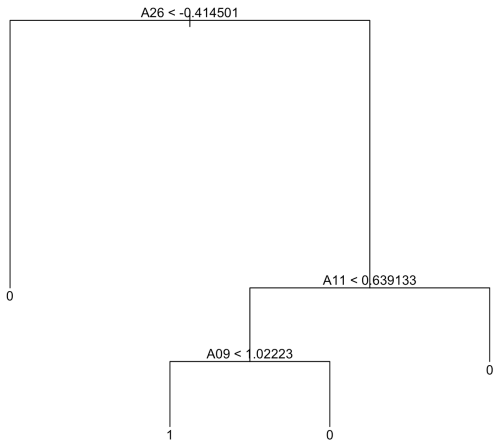


Figure 4.1: Decision tree diagram

b. Naïve Bayes:
The Naïve Bayes classifier was implemented using default settings. This probabilistic model assumes feature independence and serves as a simple yet effective baseline.

c. Bagging (Bootstrap Aggregating):
Bagging was applied using 100 trees, a commonly recommended number for initial experimentation. Bagging helps reduce variance by averaging multiple models trained on different bootstrap samples of the training data.

d. Boosting:
Boosting was also applied with 100 trees. This ensemble method builds models sequentially, where new models attempt to correct the errors of the previous one.

e. Random Forest:
The Random Forest model was run using default settings. This model combines the output of multiple decision trees trained on random subsets of the data and variables.

5. Classification Results

After training each model, predictions were made on the test dataset to compare the model performance. The performance of each model was evaluated using a confusion matrix, from which the following metrics were computed:

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	0.664	0.709	0.641	0.673
Naïve Bayes	0.626	0.495	0.655	0.564
Bagging	0.672	0.714	0.650	0.680
Boosting	0.681	0.697	0.666	0.681
Random Forest	0.717	0.734	0.701	0.717

Table 5.1: Classification result

These are the details of the metrics:

- Accuracy: The overall proportion of correct predictions of both classes.
- Precision: The proportion of predicted "Oats" that were actually "Oats". This metric indicates the reliability of positive predictions.

- Recall: The proportion of actual "Oats" correctly identified by the model. This reflects the model's sensitivity to the minority class.
- F1-score: The harmonic mean between precision and recall.

6. ROC Curve and AUC

AUC measures the likelihood that the model will rank a randomly chosen "Oats" case higher than "Other" case. It provides an overall indication of the model's ability to distinguish between the two classes. The AUC value and ROC curves for all models are as below.

Model	AUC
Decision Tree	0.684
Naive Bayes	0.698
Bagging	0.738
Boosting	0.753
Random Forest	0.782

Table 6.1: AUC values

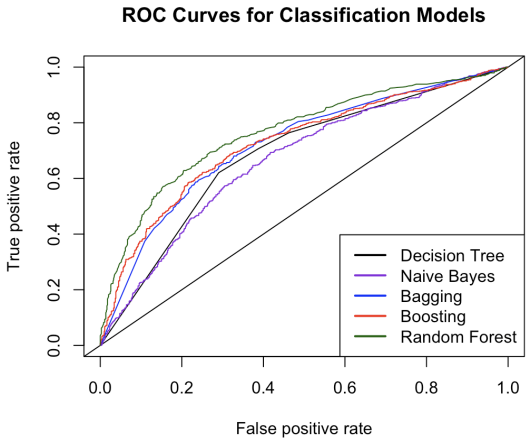


Figure 6.1: ROC Curve

7. Which Model is the Best?

To determine the most effective classification model, all classifiers were evaluated across key performance metrics. To further support the evaluation, an average score across all five metrics was calculated as shown in the comparison table below:

Model	Accuracy	Precision	Recall	F1-Score	AUC	Average
Decision Tree	0.664	0.709	0.641	0.673	0.684	0.674
Naïve Bayes	0.626	0.495	0.655	0.564	0.698	0.608
Bagging	0.672	0.714	0.650	0.680	0.738	0.685
Boosting	0.681	0.697	0.666	0.681	0.753	0.696
Random Forest	0.717	0.734	0.701	0.717	0.782	0.730

Table 7.1: Classification result summary

From the result, we notice that some classifiers are better in accuracy while some are better in other metrics. For example, Naïve Bayes demonstrates relatively low accuracy but performs reasonably well in recall and AUC, ranking third in recall and slightly outperforming the Decision Tree in AUC. Decision Tree, although weak in accuracy, but performs well in precision, surpassing those achieved by Naïve Bayes and Boosting. This highlights a critical insight: accuracy alone is not sufficient to determine the best model, especially in cases where different priorities need to be seen (e.g., prioritizing recall over precision).

Best Classifier: Random Forest
Based on the aggregated performance, Random Forest is the strongest overall model. It consistently achieves the highest values across all key metrics, demonstrating a robust balance between prediction quality and class discrimination. This makes it the most reliable classifier for the task.

Runner-Up: Boosting
Boosting performs well, especially in precision and AUC, indicating strong reliability in positive class predictions. However, its scores fall slightly less than Random Forest, placing it as a close second.

Least Effective: Naïve Bayes
While Naïve Bayes achieves a decent recall and acceptable AUC, its precision and F1-score are the lowest among all models. This suggests it produces a high number of false positives, reducing its effectiveness despite its simplicity and speed.

Although Random Forest is identified as the best overall classifier, it is important to recognize that each model exhibits distinct strengths. Depending on the specific aim, other models may be more suitable in certain contexts.

8. Attribute / Feature Importance

To gain insights into which variables most significantly influence the classification of “Oats” and “Other”, each model was examined for feature importance rankings as shown below:

Most important attributes	
Decision Tree	"A26", "A11", "A09"
Naïve Bayes	“A17”, “A26”, “A02”, “A11”, “A21”
Bagging	"A26", "A09", "A11", "A17", "A02"
Boosting	"A26", "A02", "A24", "A09", "A01"
Random Forest	“A26”, “A02”, “A24”, “A29”, “A21”

Table 8.1: Most important attributes

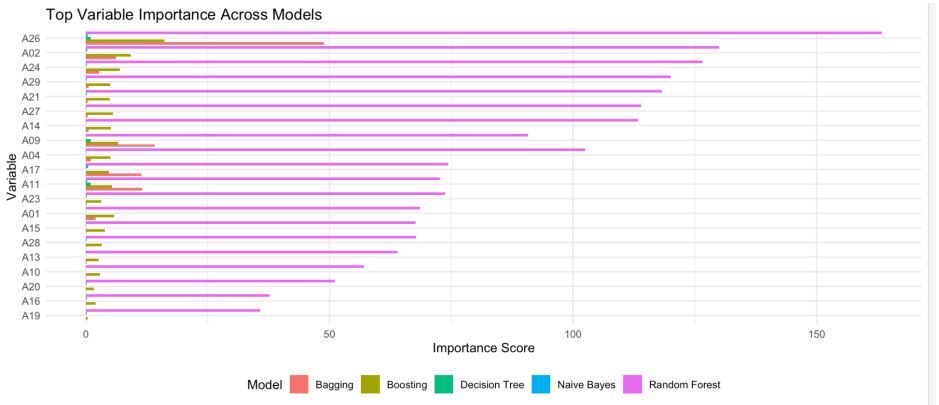


Figure 8.1: Top variable importance

A clear pattern emerges across models, with some variables identified as particularly important. First, A26 appears in all five models and ranks consistently as the most important attribute. In the Bagging model, A26 alone accounts for an importance score of 48.274, meaning that removing it could reduce model accuracy by approximately 48%. It is followed by A02 which appears in four models, indicating a strong predictive signal. A11 and A09 are present in three models, suggesting moderately strong influence on classification. These key features might be strong candidates for inclusion in simplified models, which could reduce data dimensionality and noise without sacrificing much predictive power.

Meanwhile, the least important features are:

Least important features	
Decision Tree	“A19”, “A01”, “A14”, “A15”, “A16”
Naïve Bayes	“A15”, “A04”, “A01”, “A10”, “A13”
Bagging	"A28", "A23", "A16", "A19", "A20"
Boosting	"A10", "A13", "A16", "A20", "A19"
Random Forest	“A13”, “A10”, “A20”, “A16”, “A19”

Table 9.1: Least important attributes

It is shown that A16 consistently ranked among the least important features in three models. It is also noted during earlier data exploration as having high correlation with other features (A19, A20) and numerous outliers. These characteristics further justify being omitted from the dataset. Other than that, A13, A10, A19, and A20 also rank consistently low in feature

importance. While not as clearly problematic as A16, they may be considered to be omitted when building streamlined models.

9. Justification on Model Performance Comparison

The observed variation in performance across classification models can be attributed to their underlying assumptions, algorithmic complexity, and suitability for the structure of the data. Overall, the performance differences can be explained by how well each algorithm accommodates the complexity, noise, correlation, and numerical nature of the dataset.

Naïve Bayes performed the worst among the models, which can largely be caused by its strong assumption of feature independence. In practice, this assumption is rarely satisfied. Although many features in this dataset exhibit low correlation, a few (e.g., A16, A19, and A20) are highly correlated, violating the independence assumption and degrading the model's predictive accuracy. Naïve Bayes is also relatively simplistic and does not effectively model interactions or non-linear relationships, making it less suitable for more complex datasets.

Random Forest consistently achieved the highest performance across all evaluation metrics. First, it has robustness to nonlinearity and outliers. It can capture complex, non-linear relationships and is resilient to the presence of noisy or anomalous data. Second, While correlation can be problematic for simpler models, Random Forest has the ability to mitigate high correlations. Also, due to its ensemble learning, by aggregating predictions from multiple decision trees, Random Forest reduces variance and prevents overfitting, making it highly effective on diverse and noisy datasets. The numerical nature of the data also plays to the strengths of Random Forest in identifying optimal split points in continuous data.

While the Decision Tree model is highly interpretable and performs reasonably well, it is limited when applied to noisy or complex datasets. A single tree structure may fail to capture the full pattern of relationships in the data and is prone to overfitting or underfitting. The presence of outliers and other noises likely impacted its performance.

Both Bagging and Boosting are ensemble methods aiming to improve model robustness. While these models outperform simpler approaches like Naïve Bayes and Decision Trees, they are not better than Random Forest, possibly less effective handling of feature redundancy and outliers compared to Random Forest.

10. Simple Model

Model Selection and Rationale

To develop a classifier that is interpretable and simple enough to draw by hand, I selected the Decision Tree model. This choice was due to the fact that decision trees are transparent and easy to interpret, especially when limited to a small number of decision nodes. It is also easy to be translated into a flowchart format, enabling step-by-step classification without requiring computational tools.

Attribute Selection

From the variable importance analysis in earlier models, the features A26 and A11 consistently emerged as top predictors especially for Decision Tree. These attributes were selected for the simplified model because first, they carry strong predictive power and appear across multiple models as key discriminators between "Oats"and "Other". Second, their values lend themselves well to binary threshold splitting, which simplifies decision rules and reduces model complexity.

Model Explanation

The simplified classifier uses the following rule-based logic:

- IF A26 < -0.415 → Predict Not Oats (Class = 0)
- ELSE
    - IF A11 < 0.169 → Predict Oats (Class = 1)
    - ELSE → Predict Not Oats

The diagram of the simple model is shown as below:

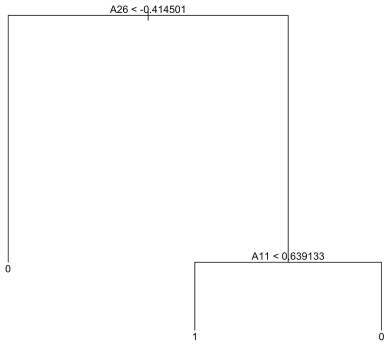


Figure 10.1: Simplified decision tree

Performance Evaluation

Using the test data from Question 3, the simplified model was evaluated using the same metrics applied earlier. Despite relying on only A26 and A11, the model performs almost similarly to the Decision Tree in Question 4. However, the precision was significantly lower than in the full model. Below is the model evaluation:

Model	Decision Tree (simplify)
Accuracy	0.661
Precision	0.613
Recall	0.666
F1-Score	0.638
AUC	0.684

Table 10.1: Simplified decision tree result summary

This simplified model highlights the strength of key features like A26 and A11 in separating classes. However, the decline in precision can be explained by the limited feature set. While A26 and A11 are powerful predictors, the exclusion of other attributes removes nuance that may help in filtering false positives, more instances being incorrectly classified as "Oats".

11. Best Tree-Based Classifier

Model Development Process

To develop the best tree-based classifier, I selected Random Forest due to its consistently strong performance in previous analyses. This model is particularly effective in handling noisy, nonlinear data and mitigating overfitting. I do several experiments with the data pre-processing and model tuning to find out the best model.

Outlier Treatment

Initially, I handled extreme values by replacing outliers using the Interquartile Range (IQR) method. This step was taken to reduce the influence of anomalies without discarding too many data points and reducing the training data size.

Feature Selection

To improve interpretability and reduce model complexity, I selected the top five most important variables identified in earlier analyses: A02, A21, A24, A26, A29, in order to only focus on the high power variables and reduce noise in the data.

Model Tuning
I set the ntree value to be 100 and I applied 10-fold cross-validation and tuned the model using the train() function in R's caret package.

Model Refinement
After testing the model specified above, the performance was unexpectedly lower than the original random forest using the full dataset, with an accuracy of only 0.651. To address this, I reverted to using the full feature set. The process began by removing a subset of variables that were either highly correlated or contributed little predictive power, which are A16, A13, A10, A19, and A20. This feature selection was informed by prior variable importance rankings and helped reduce model complexity and noise.

Next, a more refined model was built using a selected set of predictors known to carry high importance: A01 + A02 + A04 + A09 + A11 + A14 + A15 + A17 + A21 + A23 + A24 + A26 + A27 + A28 + A29.

To further improve the model, I implemented hyperparameter tuning using 10-fold cross-validation. The key hyperparameter tuned was mtry, which controls the number of variables considered at each split in the trees. A grid search was performed across a range of values (from 2 to 14), and the best combination was selected based on cross-validated performance. Additionally, the number of trees was increased to 500 to enhance stability and predictive performance.

This cross-validation approach ensured that the model was not overfitted to the training data and generalized well to unseen data.

Model Performance and Evaluation
The tuned Random Forest model achieved an accuracy of 0.716, with improvements in precision and F1-score compared to the original untuned version. The AUC value also remained high, indicating strong discriminatory power.

While the overall accuracy was more or less similar to the model built in Question 4, the increase in F1-score and precision shows that this version makes more reliable positive predictions, which is critical for classifying "Oats" correctly. This demonstrates that the tuning process and variable selection contributed to a better-balanced and more generalizable model. The complete result is as below:

Model	Tuned Random Forest
Accuracy	0.716
Precision	0.763
Recall	0.689
F1-Score	0.724
AUC	0.697

Table 11.1: Tuned random forest result summary

12. Artificial Neural Network (ANN)

Preprocessing, Feature Selection, and Training
For the ANN model, preprocessing steps were consistent with previous models to ensure comparability and model robustness:

- Data Scaling: All numerical features were standardized to ensure equal weighting during training.
- Data Split: The dataset was split into a 70% training set and 30% test set, consistent with earlier methodology.
- Feature Selection: Only the most important attribute, as identified through feature importance analysis from the five previous models, were used to reduce noise and

enhance learning performance. The selected features were: A01, A02, A09, A11, A17, A21, A24, A26, A29
- The ANN was configured with 15 hidden layers. This is the optimum value found when experimenting to capture complex non-linear relationships between variables.
- The data was also confirmed to be in numeric form, suitable for ANN processing.

Model Performance and Comparison
The ANN model outperformed all previously implemented models (Decision Tree, Naive Bayes, Bagging, Boosting, and Random Forest). It achieved 86.7% accuracy, which is the highest among all models tested and other metrics (precision, recall, and F1-score) also soars the highest. On average, the ANN's performance across all metrics was significantly higher, indicating superior generalization and predictive power.

Model	Artificial Neural Network
Accuracy	0.867
Precision	0.861
Recall	0.882
F1-Score	0.871
Average	0.870

Table 12.1: ANN result summary

Reasons for Improved Performance
The improved performance of the ANN model can be due to several factors. First, ANN models are well-suited for capturing non-linear patterns and interactions in data that simpler models might miss or oversimplify. Second, by using only the most relevant variables, the model avoids distraction from less informative features. Also, unlike Naive Bayes, which assumes independence between features, or Decision Trees, which may struggle with noisy data, ANN learns patterns directly from the data without relying on simplifying assumptions.

The ANN model demonstrates that leveraging deep learning techniques can result in marked performance improvements. Also, in this case, the use of the most important variables supports the improvement of the model.

13. New Classifier (Support Vector Machine / SVM)

Classifier and Package Used
- Classifier: Support Vector Machine (SVM)
- Package: e1071
- Package documentation:
  - https://cran.r-project.org/web/packages/e1071/
  - https://www.rdocumentation.org/packages/e1071/versions/1.7-16/topics/svm

How the Model Works
SVM classifies data by mapping it into a high-dimensional feature space where even non-linearly separable data can be divided more easily. In this space, the algorithm identifies an optimal hyperplane that separates the different classes. Once this boundary is established, new observations can be classified based on which side of the hyperplane they fall. SVM utilizes kernel functions to handle non-linear separation, where four common kernels include:
- Linear (best for linearly separable data)
- Polynomial (captures polynomial relationships)
- Radial Basis Function (RBF) (captures complex, non-linear patterns)
- Sigmoid (behaves like a neural network activation function)

Model Implementation

First, I selected the five most important features from previous models: "A01","A02", "A09", "A11", "A17", "A21", "A24", "A26", "A29". Then, I tested all four kernel types, where the RBF kernel produced the best overall results as shown below.

Kernel	Accuracy
Linear	64.8 %
Polynomial	63.6 %
Radial basis function (RBF)	66.9 %
Sigmoid	61.7 %

Table 13.1: Accuracy result for each kernel

I also did hyperparameter tuning where I used the tune() function from the e1071 package to perform a grid search over:

- cost: 0.01, 0.1, 1, 5, 10
- gamma: 0.01, 0.1, 0.5, 1

Cross-validation was used to find the optimal parameter combination.

Performance Evaluation

The model proves to have low to moderate accuracy. It outperformed Naive Bayes and Decision Tree, still much lower than top-performing models like Random Forest. However, SVM performed well in terms of AUC and ROC curve, indicating good capability in distinguishing classes although it might not be the best in exact predictions (e.g., recall or F1-score). I also took the average of the metrics above for the SVM model and the result is 0.663. This is higher than the Naive Bayes, but lower than Decision Tree. This is because the recall and F1 score of the SVM model is lower than the Decision Tree. The full result is as below:

Model	SVM
Accuracy	0.669
Precision	0.680
Recall	0.611
F1-Score	0.644
AUC	0.712

Table 13.2: Summary result for SVM

Analysis of Performance

SVM is effective in modeling complex, non-linear boundaries, especially with the RBF kernel. It outperforms simpler models like Naive Bayes and Decision Tree due to its flexibility and non-probabilistic approach. However, SVM got a relatively low recall which suggests that SVM missed many positives, possibly because the margin maximization prioritizes confidence and separation, not necessarily recall of minority class. If the decision boundary is too strict (hard margin), SVM avoids false positives at the cost of more false negatives. This also caused a lower F1 score.

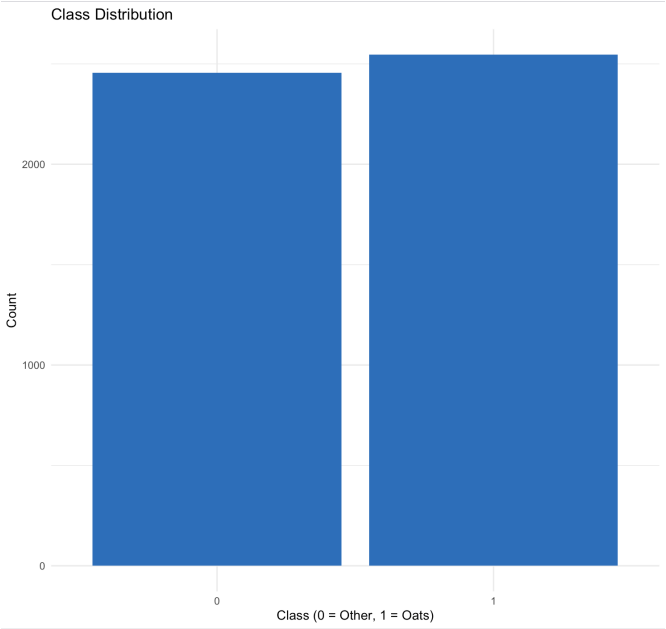
However, SVM can outperform Naive Bayes because SVM can model complex, non-linear decision boundaries (especially with kernels), while Naive Bayes is limited by its simplistic probabilistic assumptions. In terms of accuracy, it is nearly similar to Decision Trees (accuracy: 0.664), but Decision Tree is more prone to overfitting, especially on noisy or high-dimensional data, although easier to interpret.

Hence, SVM provides a robust alternative to simpler classifiers but may sacrifice recall in pursuit of precision and separation. Still, its AUC performance confirms it remains competitive in ranking and class separation tasks.

Appendix

Appendix 1: Question 1

Appendix 1.1: Distribution of the “Oats” and “Other” class



Appendix 1.2: Near-zero variances

	freqRatio	percentUnique	zeroVar	nzv
A01	102.00	7.84	FALSE	TRUE
A02	1.00	81.80	FALSE	FALSE
A04	1.01	18.76	FALSE	FALSE
A09	120.57	6.52	FALSE	TRUE
A10	515.80	39.18	FALSE	FALSE
A11	3.64	1.90	FALSE	FALSE
A13	94.77	7.46	FALSE	TRUE
A14	1.04	19.66	FALSE	FALSE
A15	417.33	28.66	FALSE	FALSE
A16	3.20	1.24	FALSE	FALSE
A17	22.89	0.78	FALSE	TRUE
A19	3.19	0.50	FALSE	FALSE
A20	3.70	0.58	FALSE	FALSE
A21	1.22	67.42	FALSE	FALSE
A23	89.50	8.92	FALSE	TRUE
A24	1.00	73.12	FALSE	FALSE
A26	1.14	84.70	FALSE	FALSE
A27	1.12	19.30	FALSE	FALSE
A28	175.43	14.48	FALSE	FALSE
A29	1.05	24.70	FALSE	FALSE

Appendix 2: Question 4

Appendix 2.1 Decision Tree

2.1.1 Confusion matrix

```
            Reference
Prediction   0   1
         0 520 213
         1 291 476
```

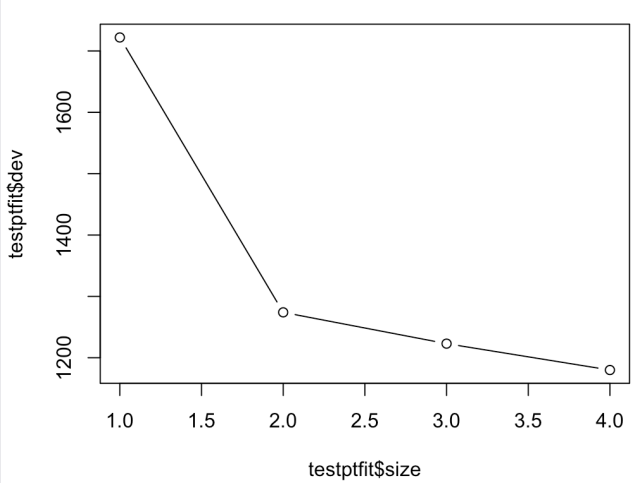
2.1.2 Performance Metrics

```
Accuracy
   0.664
> print(precision_dt)
Precision
0.7094134
> print(recall_dt)
   Recall
0.6411837
> print(f1_dt)
       F1
0.6735751
```

2.1.3 Model Summary

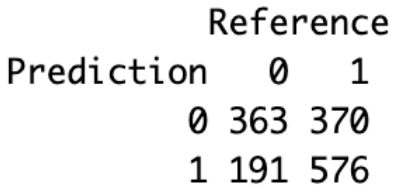
```
                Accuracy : 0.664
                  95% CI : (0.6395, 0.6879)
     No Information Rate : 0.5407
     P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.3292

  Mcnemar's Test P-Value : 0.0006039

             Sensitivity : 0.6412
             Specificity : 0.6909
          Pos Pred Value : 0.7094
          Neg Pred Value : 0.6206
              Prevalence : 0.5407
          Detection Rate : 0.3467
    Detection Prevalence : 0.4887
       Balanced Accuracy : 0.6660

       'Positive' Class : 0
```

2.1.4 Optimum Number of Terminal Nodes



2.2 Naive Bayes

2.2.1 Confusion Metrics



```
                Reference
Prediction   0   1
         0 363 370
         1 191 576
```

2.2.2 Performance Metrics

```
> print(accuracy_nb)
Accuracy
   0.626
> print(precision_nb)
Precision
0.4952251
> print(recall_nb)
   Recall
0.6552347
> print(f1_nb)
       F1
0.5641026
```

2.2.3 Model Summary

```
               Accuracy : 0.626
                 95% CI : (0.601, 0.6506)
    No Information Rate : 0.6307
    P-Value [Acc > NIR] : 0.6566

                  Kappa : 0.2475

 Mcnemar's Test P-Value : 5.684e-14

            Sensitivity : 0.6552
            Specificity : 0.6089
         Pos Pred Value : 0.4952
         Neg Pred Value : 0.7510
             Prevalence : 0.3693
         Detection Rate : 0.2420
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.6321

       'Positive' Class : 0
```

2.2.4 Apriori Probability

```
A-priori probabilities:
Y
    0     1
0.492 0.508
```

2.2.5 Conditional Probability

```
Conditional probabilities:
   A01
Y          [,1]     [,2]
  0 -0.01311099 1.008372
  1  0.01607392 0.991140

   A02
Y         [,1]      [,2]
  0  0.1522188 1.0319801
  1 -0.1400583 0.9517897

   A04
Y          [,1]      [,2]
  0 -0.01659428 0.9781895
  1  0.02479881 1.0232181

   A09
Y          [,1]     [,2]
  0  0.05538869 1.035658
  1 -0.03709388 0.962882

   A10
Y           [,1]     [,2]
  0  0.004910270 1.010504
  1 -0.004157314 1.000155

   A11
Y         [,1]      [,2]
  0  0.1148468 1.1565460
  1 -0.1296091 0.7774196

   A13
Y           [,1]      [,2]
  0 -0.002924848 1.0151615
  1 -0.003844903 0.9750113

   A14
Y          [,1]      [,2]
  0  0.03954190 1.0052083
  1 -0.03665442 0.9886234

   A15
Y          [,1]      [,2]
  0 -0.03121125 1.0241497
  1  0.01151450 0.9909279

   A16
Y          [,1]      [,2]
  0 -0.08693251 1.0248539
  1  0.08045939 0.9773006

   A17
Y         [,1]     [,2]
  0  0.2236993 1.122388
  1 -0.2084015 0.849692

   A19
Y          [,1]     [,2]
  0  0.07716479 1.006447
  1 -0.06629690 1.015635

   A20
Y          [,1]      [,2]
  0  0.09061969 0.9990700
  1 -0.08790367 0.9912401

   A21
Y         [,1]     [,2]
  0 -0.1015663 1.049298
  1  0.1048472 1.004111

   A23
Y          [,1]      [,2]
  0 -0.06518656 0.9500637
  1  0.06728077 1.0386715

   A24
Y          [,1]      [,2]
  0 -0.10068246 0.9928249
  1  0.08200653 1.0000820

   A26
Y         [,1]     [,2]
  0 -0.1620217 1.034099
  1  0.1566566 1.038632

   A27
Y          [,1]     [,2]
  0  0.05196927 1.012453
  1 -0.04388151 0.983339

   A28
Y          [,1]      [,2]
  0 -0.08140741 0.9407704
  1  0.06214457 1.0472459

   A29
Y          [,1]      [,2]
  0  0.10465610 1.0166683
  1 -0.08427187 0.9988872
```

Appendix 2.3 Bagging

2.3.1 Confusion Matrix

```
          Reference
Prediction   0   1
         0 525 208
         1 286 481
```

2.3.2 Performance Metrics

```
> print(accuracy_bg)
Accuracy
   0.672
> print(precision_bg)
Precision
0.7135061
> print(recall_bg)
   Recall
0.6496894
> print(f1_bg)
      F1
0.680104
```

2.3.3 Model Summary

```
                Accuracy : 0.6707
                  95% CI : (0.6462, 0.6944)
    No Information Rate : 0.5407
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.3425

 Mcnemar's Test P-Value : 0.0005314

            Sensitivity : 0.6473
            Specificity : 0.6981
         Pos Pred Value : 0.7162
         Neg Pred Value : 0.6271
             Prevalence : 0.5407
         Detection Rate : 0.3500
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.6727

       'Positive' Class : 0
```

Appendix 2.4 Boosting

2.4.1 Confusion Matrix

```
          Reference
Prediction   0   1
         0 511 222
         1 256 511
```

2.4.2 Performance Metrics

```
> print(accuracy_bst)
 Accuracy
0.6813333
> print(precision_bst)
Precision
0.6971351
> print(recall_bst)
   Recall
0.6662321
> print(f1_bst)
      F1
0.6813333
```

2.4.3 Model Summary

```
               Accuracy : 0.6813
                 95% CI : (0.6571, 0.7049)
    No Information Rate : 0.5113
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.363

 Mcnemar's Test P-Value : 0.1312

            Sensitivity : 0.6662
            Specificity : 0.6971
         Pos Pred Value : 0.6971
         Neg Pred Value : 0.6662
             Prevalence : 0.5113
         Detection Rate : 0.3407
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.6817

       'Positive' Class : 0
```

Appendix 2.5 Random Forest

2.5.1 Confusion Matrix

```
          Reference
Prediction   0   1
         0 538 195
         1 229 538
```

2.5.2 Performance Metrics

```
> print(accuracy_rf)
 Accuracy
0.7173333
> print(precision_rf)
Precision
  0.73397
> print(recall_rf)
   Recall
0.7014342
> print(f1_rf)
       F1
0.7173333
```

2.5.3 Model Summary

```
               Accuracy : 0.7173
                 95% CI : (0.6938, 0.74)
    No Information Rate : 0.5113
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.435

 Mcnemar's Test P-Value : 0.109

            Sensitivity : 0.7014
            Specificity : 0.7340
         Pos Pred Value : 0.7340
         Neg Pred Value : 0.7014
             Prevalence : 0.5113
         Detection Rate : 0.3587
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.7177

       'Positive' Class : 0
```

Appendix 3: Question 10

Appendix 3.1 Simplified Decision Tree

3.1.1 Confusion Matrix

```
          Reference
Prediction   0   1
         0 449 284
         1 225 542
```

3.1.2 Performance Metrics

```
> print(accuracy_simple)
 Accuracy
0.6606667
> print(precision_simple)
Precision
0.6125512
> print(recall_simple)
   Recall
0.6661721
> print(f1_simple)
       F1
0.6382374
> print(simple_auc)
[1] 0.6837184
```

3.1.3 Model Summary

```
               Accuracy : 0.6607
                 95% CI : (0.6361, 0.6846)
    No Information Rate : 0.5507
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.3198

 Mcnemar's Test P-Value : 0.01015

            Sensitivity : 0.6662
            Specificity : 0.6562
         Pos Pred Value : 0.6126
         Neg Pred Value : 0.7066
             Prevalence : 0.4493
         Detection Rate : 0.2993
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.6612

       'Positive' Class : 0
```

Appendix 4: Question 11

Appendix 4.1 Tuned Random Forest

4.1.1 Confusion Matrix

```
          Reference
Prediction   0   1
         0 559 174
         1 252 515
```

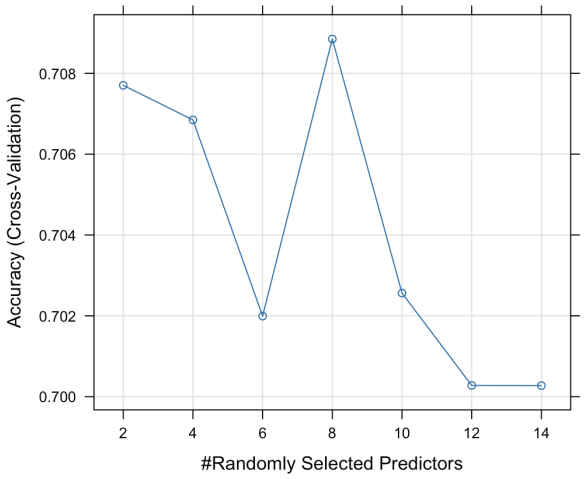
4.1.2 Performance Metrics

```
> print(accuracy_rf)
Accuracy
   0.716
> print(precision_rf)
Precision
0.7626194
> print(recall_rf)
   Recall
0.6892725
> print(f1_rf)
       F1
0.7240933
> print(rftuned_auc)
[1] 0.6971886
>
```

4.1.3 Model Summary

```
               Accuracy : 0.716
                 95% CI : (0.6924, 0.7387)
    No Information Rate : 0.5407
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.433

 Mcnemar's Test P-Value : 0.000191

            Sensitivity : 0.6893
            Specificity : 0.7475
         Pos Pred Value : 0.7626
         Neg Pred Value : 0.6714
             Prevalence : 0.5407
         Detection Rate : 0.3727
   Detection Prevalence : 0.4887
      Balanced Accuracy : 0.7184

       'Positive' Class : 0
```

4.1.4 Randomly selected predictor vs accuracy



Appendix 5: Question 13

Appendix 5.1 SVM

5.1.1. Confusion Matrix

```
          Reference
Prediction   0   1
         0 448 211
         1 285 556
```

5.1.2 Performance Metrics

```
> print(accuracy_svm)
 Accuracy
0.6693333
> print(precision_svm)
Precision
0.6798179
> print(recall_svm)
   Recall
0.6111869
> print(f1_svm)
       F1
0.6436782
> print(svm_auc)
[1] 0.7120352
```

5.1.3 Model Summary

```
               Accuracy : 0.6693
                 95% CI : (0.6449, 0.6931)
    No Information Rate : 0.5113
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3368

 Mcnemar's Test P-Value : 0.001046

            Sensitivity : 0.6112
            Specificity : 0.7249
         Pos Pred Value : 0.6798
         Neg Pred Value : 0.6611
             Prevalence : 0.4887
         Detection Rate : 0.2987
   Detection Prevalence : 0.4393
      Balanced Accuracy : 0.6680

       'Positive' Class : 0
```

5.1.4 Cross Validation Result

```
- Detailed performance results:
    cost gamma     error dispersion
1   0.01  0.01 0.4902857 0.02721211
2   0.10  0.01 0.3645714 0.03370406
3   1.00  0.01 0.3585714 0.03098533
4   5.00  0.01 0.3460000 0.03043002
5  10.00  0.01 0.3448571 0.02960205
6   0.01  0.10 0.3557143 0.02845213
7   0.10  0.10 0.3428571 0.03326524
8   1.00  0.10 0.3394286 0.02986895
9   5.00  0.10 0.3408571 0.02844576
10 10.00  0.10 0.3440000 0.03012295
11  0.01  0.50 0.4920000 0.02526234
12  0.10  0.50 0.3482857 0.02810573
13  1.00  0.50 0.3445714 0.02633122
14  5.00  0.50 0.3700000 0.02520663
15 10.00  0.50 0.3711429 0.01981548
16  0.01  1.00 0.4920000 0.02526234
17  0.10  1.00 0.3771429 0.04740906
18  1.00  1.00 0.3488571 0.02008824
19  5.00  1.00 0.3577143 0.02490071
20 10.00  1.00 0.3620000 0.02086779
```

Appendix 6: R Code

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(caret)
library(tidyverse)
library(reshape2)
library(tree)
library(e1071)
library(adabag)
library(randomForest)
library(pROC)
library(tibble)
library(rpart)
library(ROCR)
library(Metrics)
library(neuralnet)

#Question 1 and 2
rm(list = ls())
set.seed(33410712)
WD = read.csv("WinnData.csv")

#finding the proportion of 1 in the data
sum(WD$Class)/length(WD$Class)
table(WD$Class)

#balancing
WD_0 = subset(WD, Class == 0)
WD_1 = subset(WD, Class == 1)

#calculating how many more n1 sample needed
n0 = nrow(WD_0)
n1 = nrow(WD_1)
n_to_sample = n0 - n1

#oversample n1
WD_1_upsampled = WD_1[sample(nrow(WD_1), n_to_sample, replace = TRUE), ]

#combine the data
WD = rbind(WD_0, WD_1, WD_1_upsampled)
```

```
#shuffle rows
WD = WD[sample(nrow(WD)), ]
WD = WD[sample(nrow(WD), 5000, replace = FALSE),]
WD = WD[,c(sort(sample(1:30,20, replace = FALSE)), 31)]

#recheck
sum(WD$Class)/length(WD$Class)
table(WD$Class)

ggplot(WD, aes(x = as.factor(Class))) +
  geom_bar(fill = "#0073C2FF") +
  labs(title = "Class Distribution", x = "Class (0 = Other, 1 = Oats)", y =
"Count") +
  theme_minimal()

#check the data types
str(WD)

#descriptions of the independent variables
numeric_vars = sapply(WD, is.numeric)
numeric_vars
independent_vars = names(WD)[numeric_vars & names(WD) != "Class"]
independent_vars

#do a summary of statistics for each class separated
get_summary_stats <- function(data, independent_vars) {
  data.frame(
    Variable = independent_vars,
    Mean = round(sapply(data[, independent_vars], mean, na.rm = TRUE), 2),
    Median = round(sapply(data[, independent_vars], median, na.rm = TRUE),
2),
    SD = round(sapply(data[, independent_vars], sd, na.rm = TRUE), 2),
    Min = round(sapply(data[, independent_vars], min, na.rm = TRUE), 2),
    Max = round(sapply(data[, independent_vars], max, na.rm = TRUE), 2)
  )
}

WD_class0 = WD %>% filter(Class == 0)
WD_class1 = WD %>% filter(Class == 1)
summary_stats_class0 = get_summary_stats(WD_class0, independent_vars)
summary_stats_class1 = get_summary_stats(WD_class1, independent_vars)
summary_stats_class0$Class = "Class 0"
summary_stats_class1$Class = "Class 1"

summary_stats_by_class = rbind(summary_stats_class0, summary_stats_class1)
print(summary_stats_by_class)

#calculate distinct values
distinct = WD %>%
  gather() %>%
  group_by(key) %>%
  summarise(distinct_count = n_distinct(value)) %>%
arrange(desc(distinct_count))

#counting the missing values
colSums(is.na(WD))
sum(is.na(WD))

#check for near-zero variance predictors (might be omitted)
nzv_details = nearZeroVar(WD[, independent_vars], saveMetrics = TRUE)
nzv_details_rounded = nzv_details
nzv_details_rounded[] = lapply(nzv_details_rounded, function(x) {
```

```
  if (is.numeric(x)) round(x, 2) else x
})
print(nzv_details_rounded)

#identifying outliers
WD %>%
  select(all_of(independent_vars)) %>%
  gather() %>%
  group_by(key) %>%
  ggplot(aes(x = key, y = value)) +
  geom_boxplot(show.legend = FALSE) +
  facet_wrap(~ key, scales = "free") +
  labs(title = "Distribution of Predictor Variables", x = NULL, y =
"Value") +
  theme_minimal() +
  theme(
    strip.text = element_text(size = 10, face = "bold", color = "#444444"),
    plot.title = element_text(size = 14, face = "bold"),
    axis.text.x = element_blank(),
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank()
  )

#identifying highly correlated variables
cor_matrix = cor(WD[, independent_vars], use = "complete.obs")
high_cor_pairs = which(abs(cor_matrix) > 0.9 & abs(cor_matrix) < 1, arr.ind
= TRUE)
print(high_cor_pairs)
print(cor_matrix[high_cor_pairs])

#heatmap
cor_mat = cor(WD[, independent_vars])
melted_cor = melt(cor_mat)

ggplot(melted_cor, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint
= 0, limit = c(-1, 1), space = "Lab") +
  labs(title = "Correlation Matrix of Predictors", x = "", y = "") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#density plot
WD %>%
  select(all_of(independent_vars), Class) %>%
  mutate(Class = as.factor(Class)) %>%  # Ensure Class is a factor for
proper fill and legend
  pivot_longer(cols = -Class, names_to = "Variable", values_to = "Value")
%>%
  ggplot(aes(x = Value, fill = Class)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~ Variable, scales = "free", ncol = 4) +
  labs(title = "Density Plot of Predictor Variables by Class", x = "Value",
y = "Density", fill = "Class") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 14),
    strip.text = element_text(face = "bold", size = 10),
    axis.text.x = element_text(angle = 0),
    legend.position = "top"
  )
#distribution of variables between classes
```

```
WD %>%
  pivot_longer(cols = all_of(independent_vars), names_to = "Variable",
values_to = "Value") %>%
  ggplot(aes(x = as.factor(Class), y = Value, fill = as.factor(Class))) +
  geom_boxplot(outlier.alpha = 0.2) +
  facet_wrap(~ Variable, scales = "free", ncol = 4) +
  labs(title = "Boxplots of Predictor Variables by Class", x = "Class", y =
"Value") +
  scale_fill_manual(values = c("0" = "#a6cee3", "1" = "#1f78b4")) +
  theme_minimal() +
  theme(legend.position = "none")

#standardise
WD_scaled = WD
WD_scaled[, independent_vars] = scale(WD[, independent_vars])
summary(WD_scaled[, independent_vars])

#Question 3
#split train and test
set.seed(33710712)
train.row = sample(1:nrow(WD_scaled), 0.7*nrow(WD_scaled))
WD_train = WD_scaled[train.row,]
WD_test = WD_scaled[-train.row,]

WD_train$Class = as.factor(WD_train$Class)
WD_test$Class = as.factor(WD_test$Class)

#Question 4 and 5
#decision tree
ptfit = tree(Class ~., data = WD_train)
summary(ptfit)
plot(ptfit)
text(ptfit, pretty = 0)
dtpred = predict(ptfit, WD_test, type = "class")
cmdt = confusionMatrix(WD_test$Class, dtpred)
cmdt

#cross-validation
testptfit = cv.tree(ptfit, FUN = prune.misclass, K=10)
testptfit
plot(testptfit$size, testptfit$dev, type = "b")

#Best is tree with 4 nodes
prune.ptfit = prune.misclass(ptfit, best = 4)
summary(prune.ptfit)
plot(prune.ptfit)
text(prune.ptfit, pretty = 0)
dtprune.pred = predict(prune.ptfit, WD_test, type = "class")
cmdt.prune = confusionMatrix(WD_test$Class, dtprune.pred)
cmdt.prune

#calculating the accuracy, precision, recall, f1_score (decision tree)
accuracy_dt = cmdt$overall["Accuracy"]
precision_dt = cmdt$byClass["Precision"]
recall_dt = cmdt$byClass["Recall"]
f1_dt = cmdt$byClass["F1"]

print(accuracy_dt)
print(precision_dt)
print(recall_dt)
print(f1_dt)

#naive bayes
```

```
nbfit = naiveBayes(Class ~ ., data = WD_train)
nbfit
nbpred = predict(nbfit, newdata = WD_test)
cmnb = confusionMatrix(WD_test$Class, nbpred)
cmnb

#calculating the accuracy, precision, recall, f1_score (naive bayes)
accuracy_nb = cmnb$overall["Accuracy"]
precision_nb = cmnb$byClass["Precision"]
recall_nb = cmnb$byClass["Recall"]
f1_nb = cmnb$byClass["F1"]

print(accuracy_nb)
print(precision_nb)
print(recall_nb)
print(f1_nb)

#bagging
bgfit = bagging(Class ~ ., data = WD_train, mfinal=100)
bgpred = predict(bgfit, newdata = WD_test)
cmbg = confusionMatrix(WD_test$Class, as.factor(bgpred$class))
cmbg

#calculating the accuracy, precision, recall, f1_score (bagging)
accuracy_bg = cmbg$overall["Accuracy"]
precision_bg = cmbg$byClass["Precision"]
recall_bg = cmbg$byClass["Recall"]
f1_bg = cmbg$byClass["F1"]

print(accuracy_bg)
print(precision_bg)
print(recall_bg)
print(f1_bg)

#boosting
bstfit = boosting(Class ~ ., data=WD_train, mfinal=100)
bstpred = predict(bstfit, newdata = WD_test)
cmbst = confusionMatrix(WD_test$Class, as.factor(bstpred$class))
cmbst

#calculating the accuracy, precision, recall, f1_score (boosting)
accuracy_bst = cmbst$overall["Accuracy"]
precision_bst = cmbst$byClass["Precision"]
recall_bst = cmbst$byClass["Recall"]
f1_bst = cmbst$byClass["F1"]

print(accuracy_bst)
print(precision_bst)
print(recall_bst)
print(f1_bst)

#rf
rffit = randomForest(Class ~., data=WD_train)
rfpred = predict(rffit, WD_test)
cmrf = confusionMatrix(WD_test$Class, rfpred)
cmrf

#calculating the accuracy, precision, recall, f1_score (random forest)
accuracy_rf = cmrf$overall["Accuracy"]
precision_rf = cmrf$byClass["Precision"]
recall_rf = cmrf$byClass["Recall"]
f1_rf = cmrf$byClass["F1"]
```

```
print(accuracy_rf)
print(precision_rf)
print(recall_rf)
print(f1_rf)

#Question 6
#decision tree
dt_prob = predict(prune.ptfit, WD_test, type = "vector")
dt_pred = prediction(dt_prob[,2], WD_test$Class)
dt_perf = performance(dt_pred,"tpr","fpr")
plot(dt_perf, main = "ROC Curves for Classification Models")
abline(0,1)

#naive bayes
nb_prob = predict(nbfit, WD_test, type = "raw")
nb_pred = prediction(nb_prob[,2], WD_test$Class)
nb_perf = performance(nb_pred,"tpr","fpr")
plot(nb_perf, add=TRUE, col = "blueviolet")

#bagging
bg_pred = prediction(bgpred$prob[,2], WD_test$Class)
bg_perf = performance(bg_pred,"tpr","fpr")
plot(bg_perf, add=TRUE, col = "blue")

#boosting
bst_pred = prediction(bstpred$prob[,2], WD_test$Class)
bst_perf = performance(bst_pred,"tpr","fpr")
plot(bst_perf, add=TRUE, col = "red")

#random forest
rf_prob = predict(rffit, WD_test, type = "prob")
rf_pred = prediction(rf_prob[,2], WD_test$Class)
rf_perf = performance(rf_pred,"tpr","fpr")
plot(rf_perf, add=TRUE, col = "darkgreen")

legend("bottomright",
       legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest"),
       col = c("black", "blueviolet", "blue", "red", "darkgreen"),
       lwd = 2)

#extracting aucs
dt_auc = performance(dt_pred, measure = "auc")@y.values[[1]]
nb_auc = performance(nb_pred, measure = "auc")@y.values[[1]]
bg_auc = performance(bg_pred, measure = "auc")@y.values[[1]]
bst_auc = performance(bst_pred, measure = "auc")@y.values[[1]]
rf_auc = performance(rf_pred, measure = "auc")@y.values[[1]]

#comparison table
#extracting accuracy
dt_acc = confusionMatrix(WD_test$Class, dtpred)$overall["Accuracy"]
nb_acc = confusionMatrix(WD_test$Class, nbpred)$overall["Accuracy"]
bg_acc = confusionMatrix(WD_test$Class,
as.factor(bgpred$class))$overall["Accuracy"]
bst_acc = confusionMatrix(WD_test$Class,
as.factor(bstpred$class))$overall["Accuracy"]
rf_acc = confusionMatrix(WD_test$Class, rfpred)$overall["Accuracy"]

#combine into a summary table
results_summary = tibble(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest"),
  Accuracy = round(c(dt_acc, nb_acc, bg_acc, bst_acc, rf_acc), 3),
```

```
  AUC = round(c(dt_auc, nb_auc, bg_auc, bst_auc, rf_auc), 3)
)

#print the summary table
print(results_summary)

#Question 8
#decision tree
summary(ptfit)
summary(prune.ptfit)
rpart_model = rpart(Class ~ ., data = WD_train, method = "class")
importance = rpart_model$variable.importance
importance

#visualisation
dt_imp_raw = prune.ptfit$frame$var
dt_vars = table(dt_imp_raw)
dt_imp_df = data.frame(
  Variable = names(dt_vars),
  Importance = as.numeric(dt_vars),
  Model = "Decision Tree"
)

dt_imp_df = dt_imp_df[dt_imp_df$Variable != "<leaf>", ]

#naive bayes
nbfit$tables
nb_tables = nbfit$tables

#function to generate difference in mean
importance = sapply(names(nb_tables), function(var) {
  tab = nb_tables[[var]]

  if (nrow(tab) == 2 && all(rownames(tab) %in% c("0", "1"))) {
    mean0 = as.numeric(tab["0", 1])
    mean1 = as.numeric(tab["1", 1])

    return(abs(mean1 - mean0))
  }
  else {
    return(NA)
  }
})
importance = importance[!is.na(importance)]
importance_df = data.frame(Variable = names(importance), Importance =
importance
)

# Sort descending
importance_df = importance_df[order(importance_df$Importance, decreasing =
TRUE), ]
print(importance_df)

#visualisation
nb_df = importance_df
nb_df$Model = "Naive Bayes"

#boosting
names(sort(bstfit$importance, decreasing = TRUE))

#visualisation
boosting_imp = bstfit$importance
boosting_df = data.frame(
```

```
  Variable = names(boosting_imp),
  Importance = as.numeric(boosting_imp),
  Model = "Boosting"
)

#bagging
names(sort(bgfit$importance, decreasing = TRUE))

#visualisation
bagging_imp = bgfit$importance
bagging_df = data.frame(
  Variable = names(bagging_imp),
  Importance = as.numeric(bagging_imp),
  Model = "Bagging"
)

#random forest
rf_imp = importance(rffit)
#visualisation
rf_df = data.frame(
  Variable = rownames(rf_imp),
  Importance = rf_imp[, "MeanDecreaseGini"],
  Model = "Random Forest"
)

all_imp_df = rbind(nb_df, boosting_df, bagging_df, rf_df, dt_imp_df)

#plot graph of most important model
ggplot(all_imp_df, aes(x = reorder(Variable, Importance), y = Importance,
fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  labs(
    title = "Top Variable Importance Across Models",
    x = "Variable",
    y = "Importance Score"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")

#Question 10
#select the features
selected_features = c("A26", "A11", "Class")

#train a decision tree using these features
simple_tree = tree(Class ~ A26 + A11 , data = WD_train)

#visualise the decision tree
plot(simple_tree)
text(simple_tree, pretty = 0)

#predict
simple_pred = predict(simple_tree, WD_test, type = "class")
simple_cm = confusionMatrix(WD_test$Class, simple_pred)
simple_cm

#calculating the accuracy, precision, recall, F1-score (decision tree)
accuracy_simple = simple_cm$overall["Accuracy"]
precision_simple = simple_cm$byClass["Precision"]
recall_simple = simple_cm$byClass["Recall"]
f1_simple = simple_cm$byClass["F1"]
simpledt_pred = prediction(dt_prob[,2], WD_test$Class)
simple_auc = performance(simpledt_pred, measure = "auc")@y.values[[1]]
```

```
#print
print(accuracy_simple)
print(precision_simple)
print(recall_simple)
print(f1_simple)
print(simple_auc)

#Question 11
#select random forest as the best model
rffit_selected = randomForest(Class ~., data=WD_train, ntree=100)
rfpred_selected = predict(rffit_selected, newdata = WD_test)
cmrf_selected = confusionMatrix(WD_test$Class, rfpred_selected)

cmrf_selected

#tuning
#10-fold cross-validation
ctrl = trainControl(method = "cv", number = 10)

tunegrid = expand.grid(mtry = c(2, 4, 6, 8, 10, 12, 14))

#selected variables
selected_vars = c("A01", "A02", "A04", "A09", "A11", "A14", "A15", "A17",
"A21", "A23", "A24", "A26", "A27", "A28", "A29")

#start tuning
set.seed(333410712)
rf_tuned = train(Class ~ A01 + A02 + A04 + A09 + A11 + A14 + A15 + A17 +
A21 + A23 + A24 + A26 + A27 + A28 + A29,
                 data = WD_train,
                 method = "rf",
                 trControl = ctrl,
                 tuneGrid = tunegrid,
                 ntree = 500)

print(rf_tuned)
plot(rf_tuned)
rfpred_tuned = predict(rf_tuned, newdata = WD_test[, selected_vars])
cmrf_tuned = confusionMatrix(WD_test$Class, rfpred_tuned)
cmrf_tuned

#calculating metrics after tuning
accuracy_rf = cmrf_tuned$overall["Accuracy"]
precision_rf = cmrf_tuned$byClass["Precision"]
recall_rf = cmrf_tuned$byClass["Recall"]
f1_rf = cmrf_tuned$byClass["F1"]
rftuned_prob = predict(rf_tuned, WD_test[, selected_vars], type = "prob")
rftuned_pred = prediction(rftuned_prob[,2], WD_test[, selected_vars]$Class)
rftuned_perf = performance(rftuned_pred,"tpr","fpr")
rftuned_auc = performance(rftuned_pred, measure = "auc")@y.values[[1]]


print(accuracy_rf)
print(precision_rf)
print(recall_rf)
print(f1_rf)
print(rftuned_auc)

#Question 12
WD_train_nn = WD_train
WD_test_nn = WD_test
WD_train_nn$Class = as.numeric(as.character(WD_train_nn$Class))
```

```
WD_test_nn$Class = as.numeric(as.character(WD_test_nn$Class))
selected_vars = c("A01","A02", "A09", "A11", "A17", "A21", "A24", "A26",
"A29")

annfit = neuralnet(Class ~ A01 + A02 + A09 + A11 + A17 + A21 + A24 + A26 +
A29, WD_test_nn, hidden=15, linear.output = FALSE)
annpred = compute(annfit, WD_test_nn[, selected_vars])
prob = annpred$net.result
pred = ifelse(prob>0.5, 1, 0)

pred_factor = factor(pred, levels = c(0, 1))
obs_factor = factor(WD_test_nn$Class, levels = c(0, 1))
cmann = confusionMatrix(pred_factor, obs_factor)
cmann

#calculate metrics
accuracy_ann = cmann$overall["Accuracy"]
precision_ann = cmann$byClass["Precision"]
recall_ann = cmann$byClass["Recall"]
f1_ann = cmann$byClass["F1"]

print(accuracy_ann)
print(precision_ann)
print(recall_ann)
print(f1_ann)

#Question 13
WD_train_svm = WD_train
WD_test_svm = WD_test
WD_train_svm$Class = as.factor(WD_train_svm$Class)
WD_test_svm$Class = as.factor(WD_test_svm$Class)

svmfit = svm(Class ~., data = WD_train_svm, kernel = "radial")
svmpred = predict(svmfit, newdata = WD_test_svm)

cm = table(Predicted = svmpred, Actual = WD_test_svm$Class)
accuracy = sum(diag(cm)) / sum(cm)

print(cm)
cat("Accuracy:", round(accuracy * 100, 2), "%\n")

#tuning
selected_vars = c("A01","A02", "A09", "A11", "A17", "A21", "A24", "A26",
"A29")
svmfit = svm(Class ~ A01 + A02 + A09 + A11 + A17 + A21 + A24 + A26 + A29,
             data = WD_train_svm,
             kernel = "radial",
             probability = TRUE)
svmpred = predict(svmfit, newdata = WD_test_svm[, selected_vars])

tuned_svm = tune(svm, Class ~ A01 + A02 + A09 + A11 + A17 + A21 + A24 + A26
+ A29, data = WD_train_svm, kernel = "radial",
ranges = list(
  cost = c(0.01, 0.1, 1, 5, 10),
  gamma = c(0.01, 0.1, 0.5, 1)
),
probability = TRUE)

summary(tuned_svm)

#best tuned model
best_svm = tuned_svm$best.model
```

```
#predict using the best model
svmpred_best = predict(best_svm, newdata = WD_test_svm[, selected_vars])
pred_factor = factor(svmpred_best, levels = c(0, 1))
obs_factor = factor(WD_test_svm$Class, levels = c(0, 1))
cmsvm = confusionMatrix(pred_factor, obs_factor)

cmsvm

#calculate metrics
accuracy_svm = cmsvm$overall["Accuracy"]
precision_svm = cmsvm$byClass["Precision"]
recall_svm = cmsvm$byClass["Recall"]
f1_svm = cmsvm$byClass["F1"]

svm_prob = predict(best_svm, newdata = WD_test_svm[, selected_vars],
probability = TRUE)
svm_prob_values = attr(svm_prob, "probabilities")[, "1"]
svm_pred = prediction(svm_prob_values, WD_test_svm$Class)
svm_perf = performance(svm_pred, "tpr", "fpr")
svm_auc = performance(svm_pred, measure = "auc")@y.values[[1]]

print(accuracy_svm)
print(precision_svm)
print(recall_svm)
print(f1_svm)
print(svm_auc)
```