Analytics, Assignment 1

Author(s): Petersen Trentin (PTRTRE004)
Stevenson Daniela (STVDAN011)

August 13, 2024

Question 1

(a)

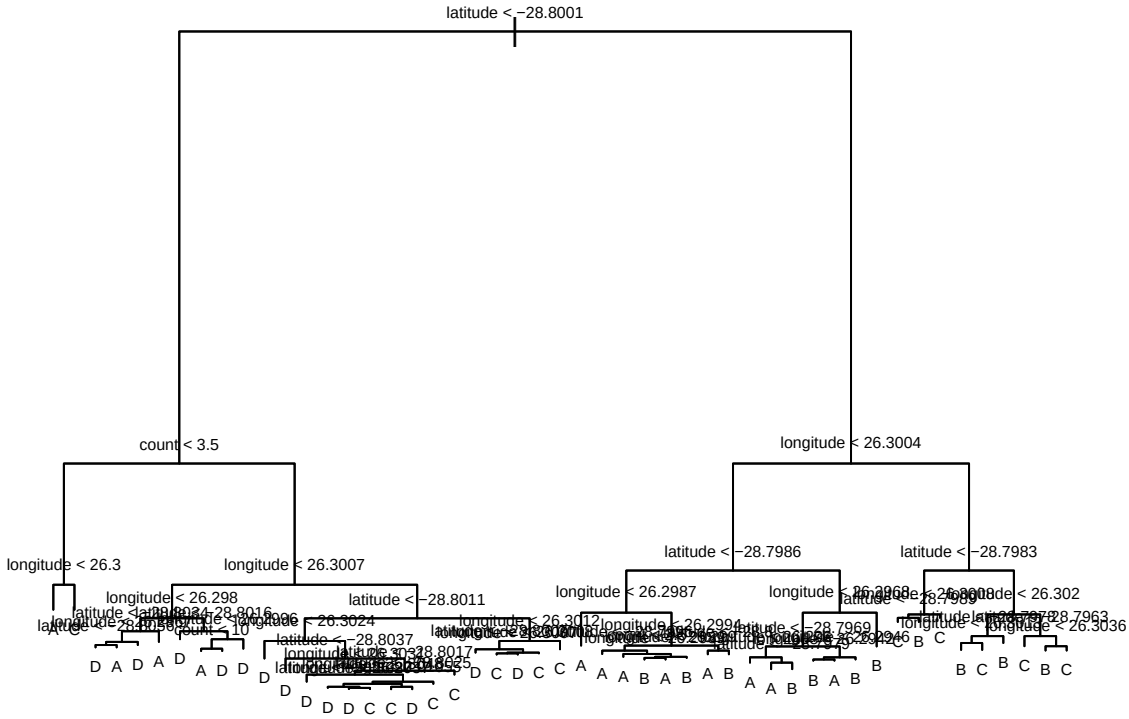

Figure 1: Overfitted tree fitted to Q1dat (Model1).

Firstly, the 'Q1dat' dataset was split into an 80/20 train/test split. Figure 1 displays the overfitted classification tree that was fit to the training data, call it Model 1. This tree has a very high complexity, with 41 terminal nodes. Furthermore if we look at the frame component of the tree object we can observe what variables were used to determine where the splits were performed in the tree. The latitude and longitude variables were observed to be of the most importance when determining the splits. The variables: Pests, Counts and Height can therefore be deemed of little importance when building a tree to determine the final mealie quality.

Most notably, each node of this overfitted tree was homogeneous. This means there were only (n) observations of one class (A,B,C or D) in each node. This was achieved through removing the stopping criterion, which was done by setting the 'tree.control' parameters *minsize* and *mindev* equal to 2 and 0 respectively. This means only 2 observations were required to be in a node for a split to be performed. Additionally, no reduction in the deviance was required for a split to be performed, hence everything was so clustered as the change in deviance was proportional to the branch length. Therefore, splits were only allowed to occur until either a terminal node holds a single observation or multiple observations of the same class.

The homogeneity of the terminal nodes was also validated by extracting the leaves/terminal nodes from the frame of the tree object, extracting a matrix of the different probabilities of each class within each node, and checking if that matrix only included 0's and 1's (which implies that each node only holds one class and is homogeneous). The code for this can be found in (1) of the appendix.

Model 1 Testing Misclassification Rate = 0.04

(b)

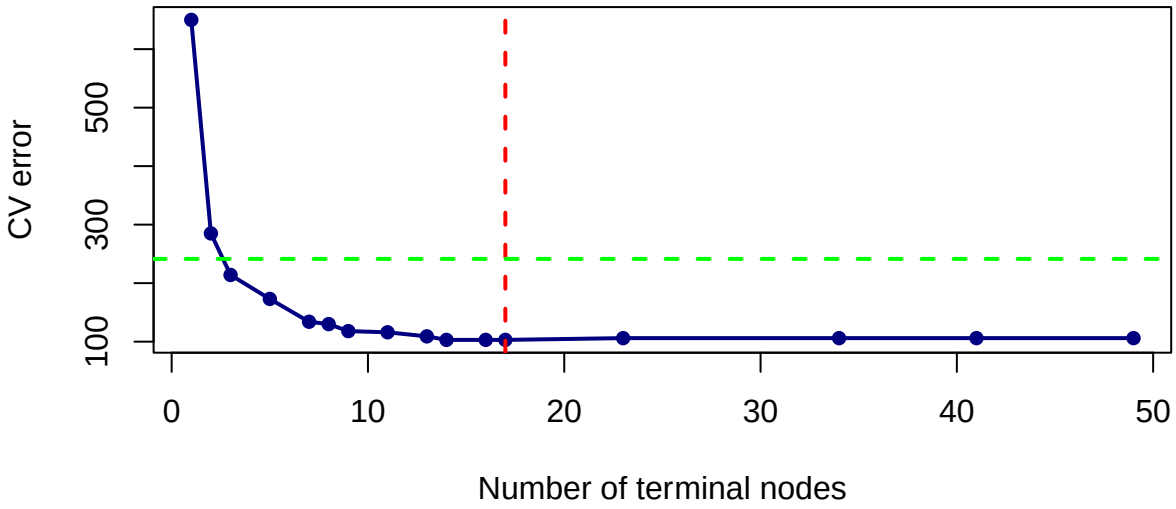

Figure 2: Cross-validated pruning results on the overfitted tree fitted to Q1dat.

Figure 2 indicates the size of the tree corresponding to one standard error above the mean (green, horizontal) is very small with a relatively large CV error. Therefore, in this context it seems more logical to choose a tree size that corresponds to the minimum CV error, which in this case is a tree with 17 terminal nodes, call it Model 2. The overfitted tree pruned to 17 terminal nodes gives us:

Table 1: Misclassification Rates of Models 1-2.

Model 2 Train	Model 2 Test
0.053	0.08

Table 1 shows the overfitted tree pruned to 17 terminal nodes having a testing misclassification rate which is higher than both its training misclassification rate and Model 1's testing misclassification rate.

Thorugh greater observation, it was found that as the tree was pruned less (the size of the pruned model approached the size of the overfitted model) the testing misclassification rate decreased. This may have been as a result of the pruned model underfitting to the data, which may have very complex underlying patterns which are better picked up by the overfitted model leading to the pruned model performing worse than the overfitted model.

(c)

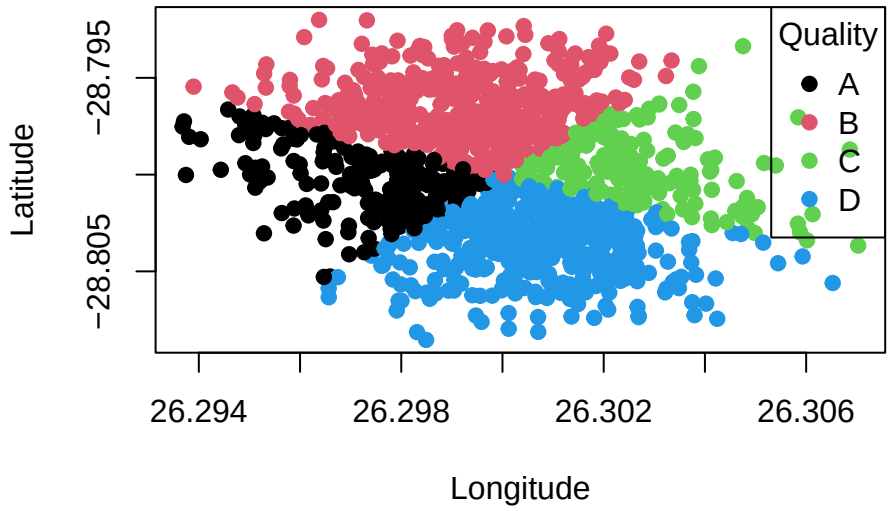

Figure 3: Plot of the quality of mealie against location data (latitude and longitude).

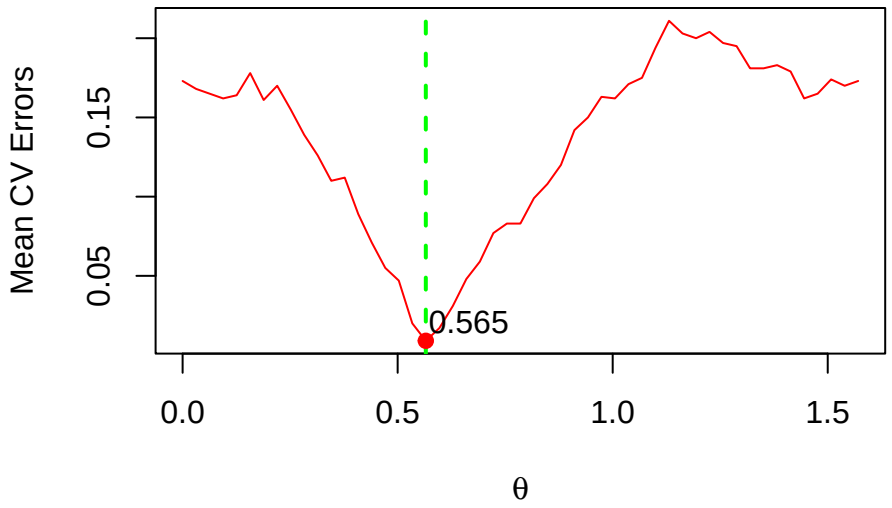

Figure 4: MSE for 10-Fold CV against theta.

The theta value used to rotate the latitude and longitude data that resulted in the lowest cross-validated error, as seen in Figure 4 was:

$$\theta = 0.565$$

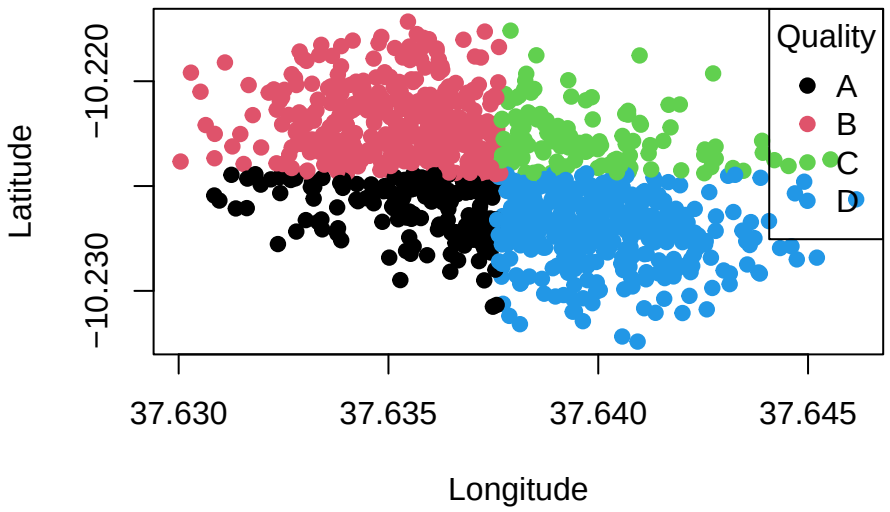

Figure 5: Plot of the quality of mealie against rotated location data (latitude and longitude).

$\theta = 0.565$ was then used to rotate the latitude and longitude components of the data such that the decision boundaries were orthogonal to the axes as shown above in Figure 5.

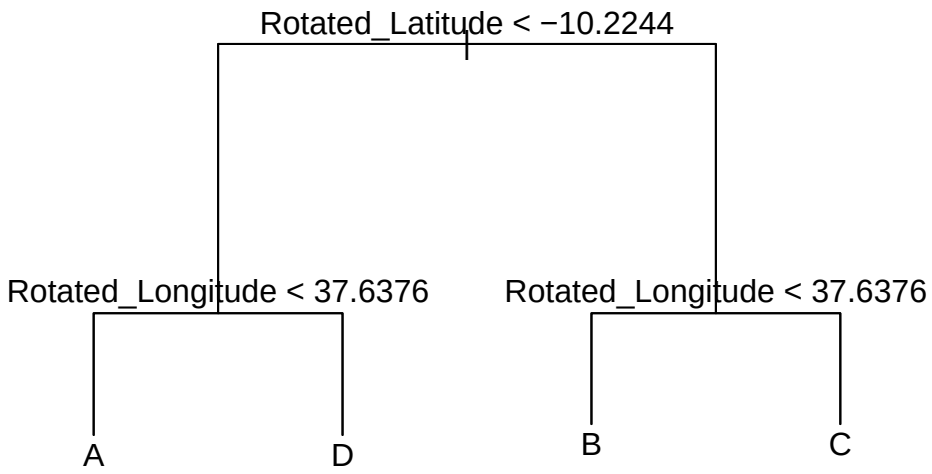

Figure 6: Tree fitted to Q1dat with the location data rotated and then pruned (Model 3).

Figure 6 shows the tree that was fitted to the rotated data and then pruned to have 7 nodes (4 terminal nodes and 3 internal nodes including the root node) and a depth of 2, call it Model 3. Once again, when inspecting the frame of the tree object, only the Latitude and Longitude variables were used to create splits. Thus, like in the overfitted tree, the other predictor variables were redundant in the determining of splits.

Table 2: Testing Misclassification Rates of Models 1-3.

Model 1	Model 2	Model 3
0.04	0.08	0.008

The testing misclassification error for Model 3 was lower than the testing misclassification error for both Model 1 and Model 2. So even though Model 3 only has 4 terminal nodes, with a depth of 2, it does an excellent job at predicting on the test data; provided that a rotation of $\theta = 0.565$ is applied to the latitude

and longitude variables in the data with which it is using to make predictions. This makes sense as we can split the rotated data near perfectly into four quadrants, representing the four different quality classes.

Question 2

(a)

Initially, testing was done to determine whether to use the lambda which yielded the minimum cross-validation error (lambda.min), or to use the lambda which lies one standard error above the minimum cross-validation error (lambda.1se). This was done by keeping $\alpha = 0.5$ and calculating the test accuracy for the model using both lambda.min and lambda.1se for 10 different seeds, giving us 10 different test accuracy measurements. The test accuracy measurements were then aggregated, producing the values seen in Table 3 below.

The difference in the test accuracy of a model using 'lambda.1se' to a model using 'lambda.min' is negligible, therefore, it makes sense to choose the lambda value which will yield a model with a lower complexity, which will be lambda.1se.

Table 3: Test Accuracy of Elastic-Net Classification Model using Lambda.min/Lambda.1se.

	Lambda Min	Lambda 1se
Test Accuracy	0.656	0.66

Additionally, the decision to remove the variables 'Defending.Team' and 'Chasing.Team' was made. This was determined in a similar manner to the lambda value situation. The out-of-sample accuracy was calculated 10 times using 10 different seeds and then averaged for 2 different models.

Two models were fit, one including the two aforementioned variables and another model not containing the two variables. Table 4 shows that the difference in test accuracy is negligible. This means the obvious choice is that of the simpler model, i.e. the model excluding the 'Defending.Team' and 'Chasing.Team' variables as our final model.

Table 4: Test Accuracy of the models Including/Excluding the Defending.Team and Chasing.Team variables.

	Include Teams	Exclude Teams
Test Accuracy	0.664	0.648

Moreover, another reason to exclude the 'Defending.Team' and 'Chasing.Team' variables was the volatile nature of sports teams where player compositions change frequently. The data of the 743 matches comes from several iterations of the annual IPL tournament, so it is almost guaranteed that the players in each team were not always the same across different matches. Therefore, using them to predict future outcomes is meaningless, as year to year the experience and skill levels in a team shift. This means that even though they play under the same team name as before, it does not implicitly suggest that that the team's performance will remain consistent in following years.

A custom range of λ was used as it allowed for the granularity of the λ to be more finely controlled. The range used was: $10^{-4} \leq \lambda \leq 1$ as any $\lambda > 1$ suggested that an empty model be used with a higher relative cv-error.

After the range of λ was determined a sequence of alpha values, $0 \leq \alpha \leq 1$, was iterated over, each time performing 9-Fold cross-validation to obtain a cross-validation misclassification error for the model. The lowest misclassification error and its corresponding α value were stored. This is how the optimal α value was determined as it, in combination with the range of λ, produced the model with the lowest misclassification rate.

The parameters used in the regularisation for the final model:

$$\alpha = 0.19 \quad and \quad \lambda = 0.26$$

Which resulted in the the final model with the coefficients shown in Table 5 below.

Table 5: Elastic-net logistic regression model fitted to Q2dat.

	estimate
(Intercept)	-0.225
Defending.StadiumAWAY	0.000
Defending.StadiumHOME	0.000
Defending.StadiumNEUTRAL	0.000
TimeDay	0.000
TimeDay/Night	0.000
Defending.TossLOST	0.000
Defending.TossWON	0.000
Bat2.Strength	0.000
Bowl2.Strength	0.000
First.Inn.Score	0.336
First.Inn.Median	0.000
Second.Inn.Median	0.000
Ground.Altitude	0.000
Ground.Length	0.000
Ground.Width	0.000

The final model equation can be written as follows:

$$logit(Defending.Result) = -0.225 + 0.336 * First.Inn.Score \qquad (eqn.1)$$

From equation 1, the log odds of the defending team winning after the first innings is -0.225 when $First.Inn.Score = 0$. It is also shown that a one unit increase in $First.Inn.Score$ will lead to a 0.336 increase in the log odds of the defending team winning after the first innings.

Table 6: Odds effects for the final elastic-net logistic regression model fitted to Q2dat.

Coefficient	e^{B_j}
(Intercept)	0.799
First.Inn.Score	1.399

This can also be interpreted as the odds of the defending team winning after the first innings being $e^{-0.255} = 0.799$ when the $First.Inn.Score = 0$. And for a one unit increase in the $First.Inn.Score$ the odds of the defending team winning after the first innings will change by a factor of $e^{0.336}$. This is represented by Table 6 above.

Approaching this from a cricket point of view, it makes sense that the $First.Inn.Score$ was the most important predictor of whether or not the defending team will win, because if the team who bats first (the defending team) manages to put up a large score then the chasing team has a large score to chase. This could have a significant psychological impact, potentially impacting their performance and ability to match the first innings score and could increase the defending teams likelihood of winning in general.

The testing accurracy and the F_1-score can be seen in Table 7 below.

Table 7: Testing Accuracy and F_1-score for the final model using α and λ from Table 5.

Testing Accuracy	F_1-score
0.685	0.472

(b)

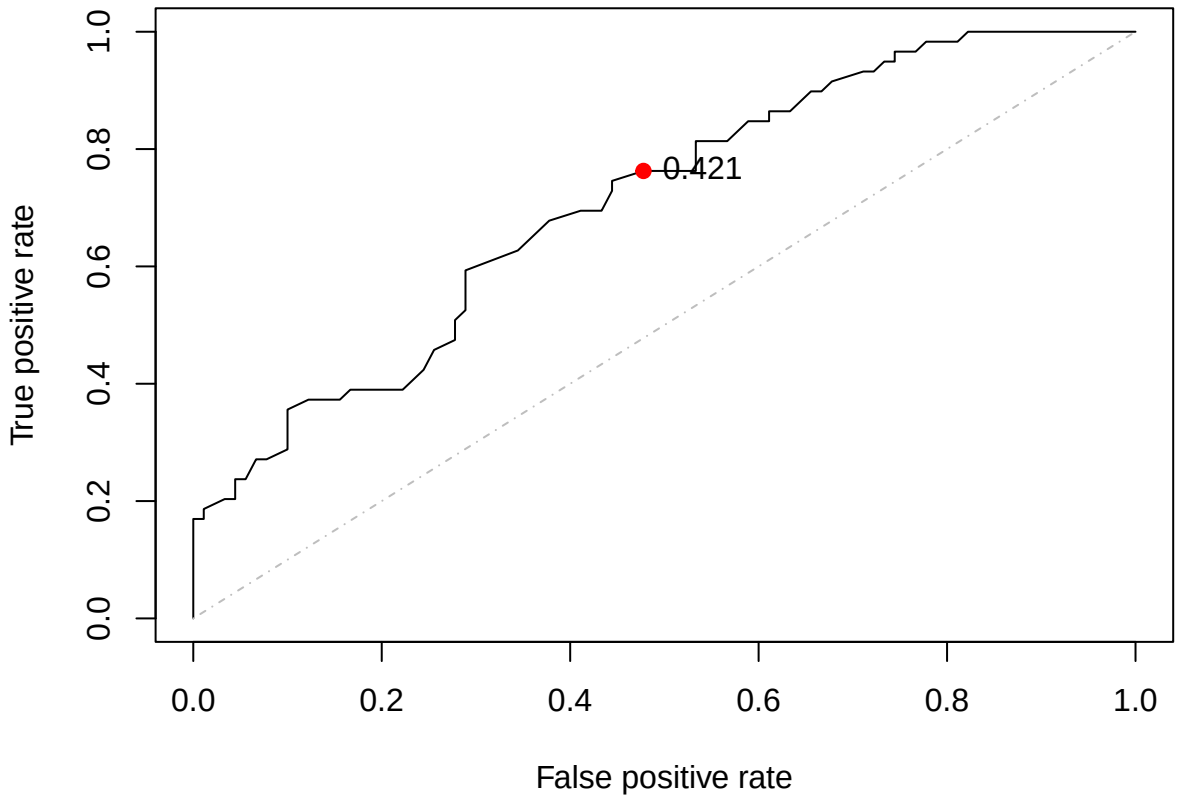

Figure 7: ROC Curve for the final logistic regression model, regularised using an elastic-net approach.

The final model was used to predict outcomes using out-of-sample data, these predictions were then used to plot an ROC curve shown in Figure 6. The area under the ROC curve in Figure 6 was calculated to be 0.708.

The minimum decision rule threshold, τ, that resulted in a recall of at least 0.75 was then determined to be 0.421 as marked by the red point in Figure 6 above.

This was validated by applying this τ decision rule to predictions made on the test data and then calculating the recall. This is demonstrated in the code and output provided in (2) of the appendix.

Exploratory Analysis

The data provided contains information pertaining to patients with early-stage Parkinson's disease. There are eighteen explanatory variables, seventeen numeric and one binary (the sex of the patient - male/female). The goal of the exercise is to predict the UPDRS (response) through the use and implementation of various models. Before model implementation, an exploratory analysis will be conducted.

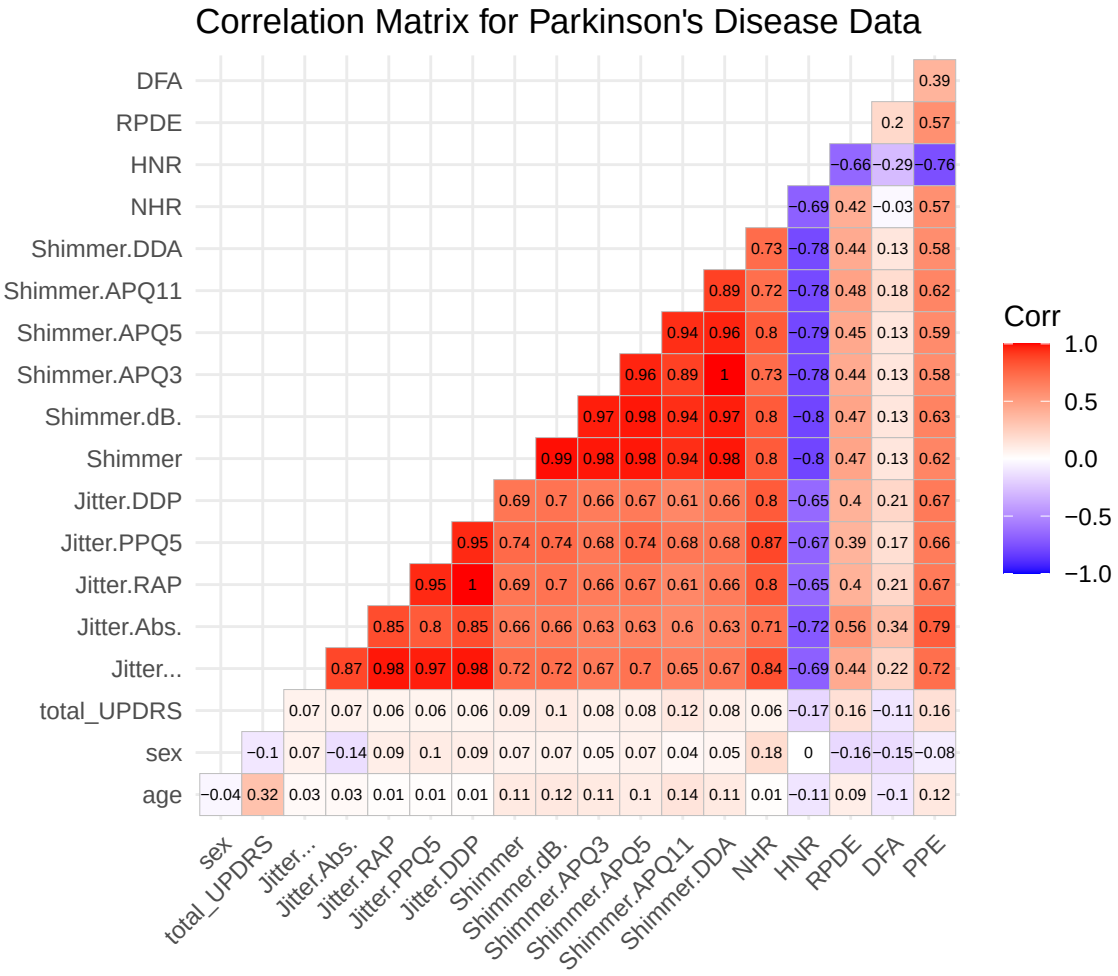

Figure 8: Heat Map Displaying the Correlation within the Parkinson's Disease Data

As seen in the correlation plot above, the Parkinson's Disease data contains variables that are highly correlated. Majority of the correlation coefficients have strong positive correlation coefficients (above 0.6), with a large percentage very strong and above 0.8. Notably, two relationships present are perfectly positively correlated with a correlation coefficient of 1. These relationships are between the variables Shimmer.APQ3 and Shimmer.DDA, and Jitter.Rap and Jitter.DDP.

The first step was splitting the data (80/20) into training and test data respectively. Each model was chosen based on their respective optimal hyperparameter selections.The comparison of each model was based on how well it predicts the UPDRS by evaluating their out-of-sample RMSE. Lower RMSE implies greater predicitve accuracy.

Linear Model with Variable Selection/Regularisation Applied

The aim of this exercise is response prediction, not variable interpretation. Thus, all explanatory variables, including sex, were standardized for both the training and test data, equalising the scales of all predictors.

In order to find the most accurately performing linear model with variable selection and regularisation, the hyperparameters alpha (relative weight of the penalties for ridge and lasso) and lambda (strength of the regularisation) were tuned. It allows the model to find the optimal combination that forms a regularised linear model that most accurately predicts the UPDRS and thus has the lowest RMSE. The tuning utilised a grid containing a multitude of alpha values between 0 and 1, and 100 lambda values between 10^{-4} and 10^{5}.

The model also utilised 10-fold cross validation to prevent the model from overfitting to the training data when selecting the optimal hyperparameters for the model.

Table 8: Comparison of RMSE Across Differnt Linear Models

Unregularised Model	Ridge ($\alpha = 0$)	Grid Search Best Model ($\alpha = 0.002$)	Elastic Net ($\alpha = 0.5$)	Lasso ($\alpha = 1$)
11.8043	10.0908	10.0894	10.0926	10.0935

As seen in the table, the linear model produced with the grid search performed best as it had the lowest RMSE value (of 10.893), though only marginally. Thus, the best linear model with regularisation had the hyperparameters: $\alpha = 0.002$ and $\lambda = 0.231$. This α value is very small - alsmost 0 and thus, very close to a ridge regression.

Ridge regression, and regularisation, was first developed to handle data with highly correlated explanatory variables. As previously discussed, the Parkinson's data is highly correlated, indicating a model with a form regularisation may be useful.Ridge regression gradually decays the coefficients (slow shrinkage), where as lasso performs variable selection by shrinking the coefficients to zero. In this case, the linear model performs best through approximately a ridge regression, though it does have some aspects of lasso.

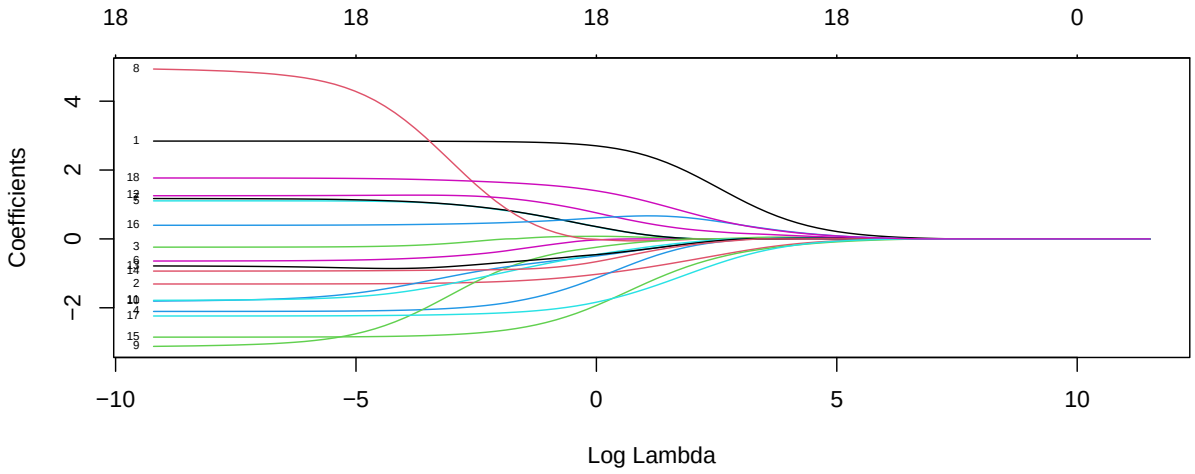

Figure 9: Coefficient profiles for the Linear Model with $\alpha = 0$

The illlustration above shows how the coefficients change as log lambda increases. The numbers labelled at the top indicate the amount of non-zero coefficiets, and because the number decreases it indicates this regularisation performs some variable selection, much like a Lasso model. Further, this is unlike Ridge which undergoes no variable selection, though it does gradually decay to zero which is an aspect of Ridge. Thus, it can be argued the chosen Linear Model has is a type of elastic net and is a combination of ridge and lasso.

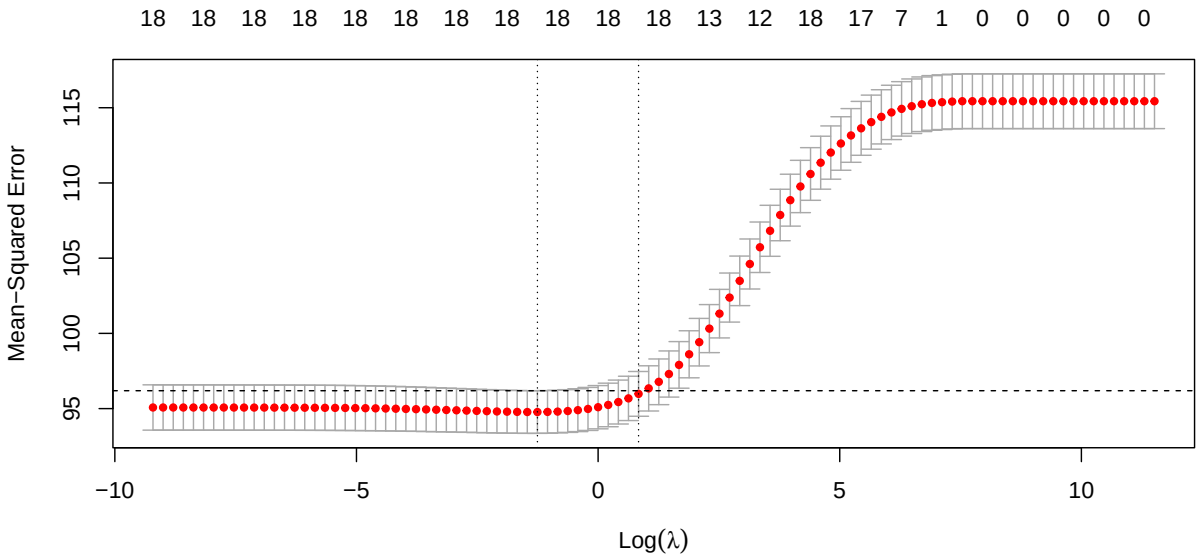

Figure 10: 10-fold CV MSEs as a function of $\log(\lambda)$ for the Linear Model with $\alpha = 0.002$

This figure illustrates the tuning parameter, λ, yields the minimum CV error (red dots) when $\lambda = 0.231$.

K-Nearest Neighbours (KNN)

KNN models require the explanatory variables to be standardized as it is a distance based model. KNN forms new observations based on predicted values formed when it chooses and groups the 'K' closest points, and uses them to predict a value that forms a new observation.

Thus, the distance between points is affected by the scales of the variables. Standardization ensures all explanatory factors contribute equally to the distance measurements.

The hyperparameter associated with KNN is number of neighbours, K, the model chooses to examine. K is user-defined. For very low values of K (e.g. K = 1), the model tends to overfit to the training data as fits to its noise and outliers. This will lead to a higher error rate (e.g. the RMSE) on the test set as its 'test' response predictions are less accurate (large error between predicted and actual values for UPDRS).

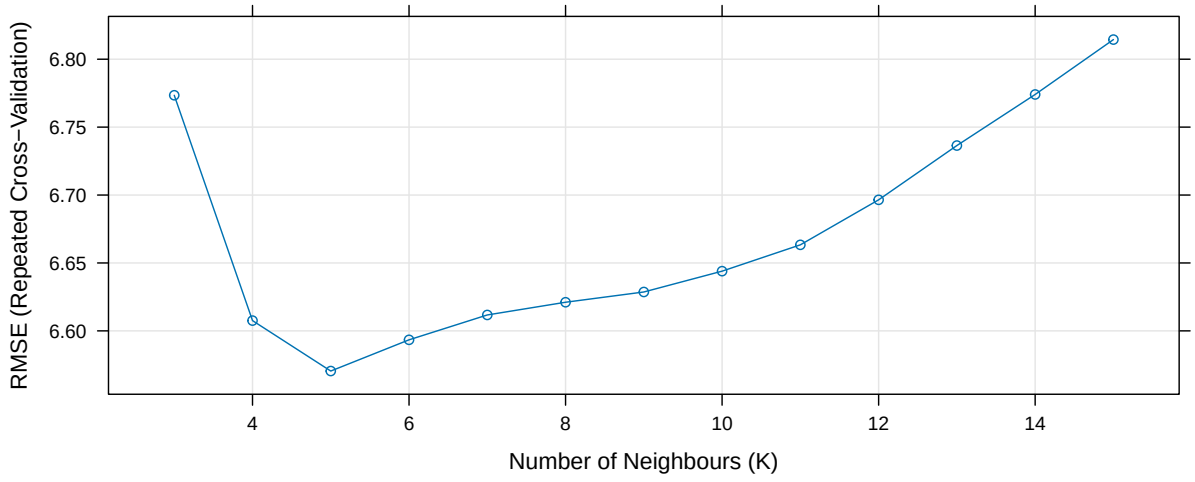

Figure 11: Repeated CV results for KNN

The figure above indicates the KNN model with the lowest corresponding RMSE according to the CV procedure, is one where its hyperparameter is $K = 5$. As K decreases, the model flexibility increases. Thus, for any $K < 5$ the model will overfit to the data, though the cross-validation procedure helps prevents this from occurring.

Random Forest

Random Forest is an ensemble modelling technique that reaches a final output by constructing multiple decision trees, a 'forest', and combining their results. Random Forest combats the weakness of high sampling variability found in decision trees. This weakness is a result of how Decision Trees are prone to overfitting to the training data. Random Forest is a more robust method and avoids overfitting by having a marginal difference in the way it trains each of its trees. Additionally, standardisation was not necessary because ensemble techniques like Bagging, Boosting and Random Forest are not sensitive to the magnitude of explanatory variables.

The tuned RF model utilised the 'ranger' and 'train' functions to best select the optimal hyperparameters. The first hyperparameter is 'mtry', indicating the number of random variables at each split that are to be randomly sampled candidates. The default is a third of the variables but the tuned model used a grid that move between 10 and 18 (the number of predictors). Next is 'splitrule' which controls how nodes are split in the trees. The tuned model used "variance" as the splitting rule. Lastly, 'min.node.size' indicates the smallest size of a node. The default is 5 and the tuned model moves between 1 and 10. *The number of trees is not considered a hyperparameter in 'ranger', however the chosen amount for each model was 500.

To prevent overfitting when parameter tuning for RF, the trained model was controlled using the "out-of-bag (OOB)" error method.

Table 9: Comparison of RMSE Across Differnt Random Forest Models

Bagging Model	Standard RF Model	RF using Ranger Model	Ranger with Grid Search Model
3.2022	4.8003	3.1944	3.1936

The table above depicts the RF model with the lowest RMSE (most accurate response predictions) was the RF model utilising a grid to tune its hyperparameters. It is important to note that though it had the lowest RMSE, 3.1936, it is still very close to the Bagging Model and the RF using 'ranger' without a tuning grid. This is largely because the grid-tuned model, like the Bagging and Ranger, also had its hyperparameter 'mtry' equal eighteen (the total number of predictors).

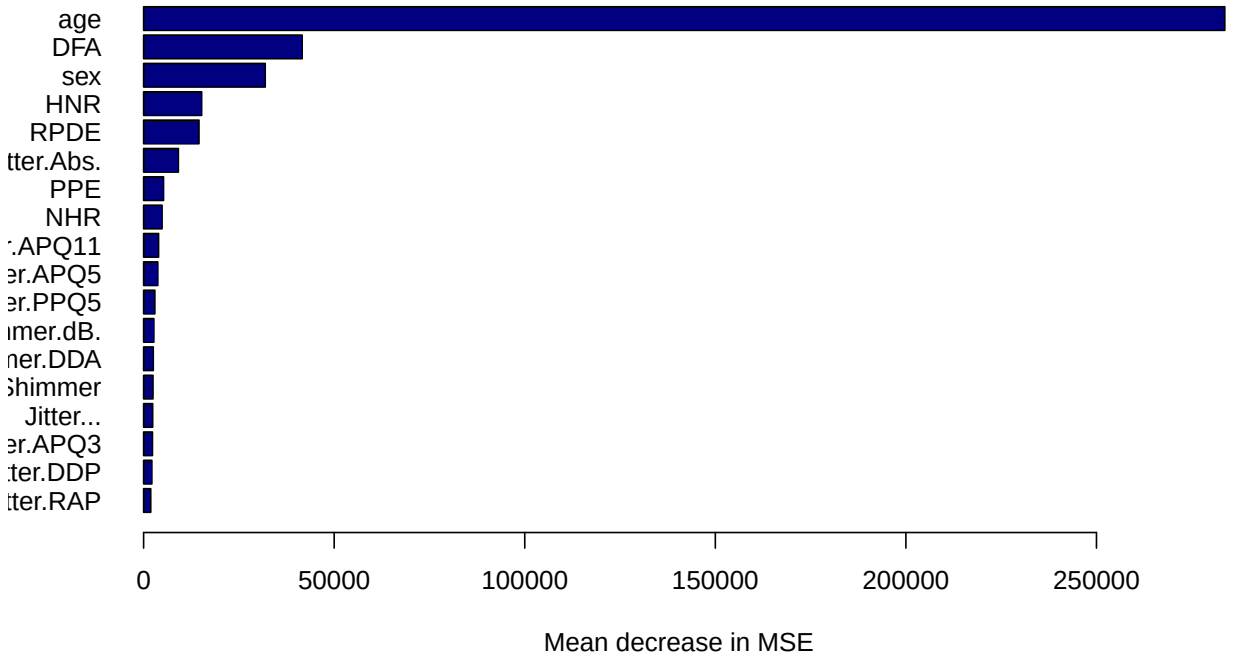

Figure 12: Variable Importance Plot for Grid Search RF model

As seen in the plot above, the explanatory variable with the most significant effect on the model's RMSE (and subsequently its predictions) is age as it has the longest corresponding bar. Thus, age produced the greatest average reduction in the RMSE. Age is then followed by DFA and sex respectively, in terms of its impact on the model, however age by far has the greatest influence and variable importance.

Boosted Tree Model

Gradient Boosting is another ensemble technique like Random Forest; however it has a higher accuracy because the algorithm is formed to progressively learn from its mistakes. Gradient Boosting optimises its ensemble by using gradient descent.

The Gradient Boosted models also undergo tuning to best select the optimal hyperparameters. These hyperparameters are tuned by creating a Gradient Boosting Machine (GBM) tuning grid. The first hyperparameter is 'n.trees', representing the total amount of trees to fit. Increasing the number of trees to fit will minimise the RMSE, however it also increases the computational time. In this case, models tuned the number between 500 and 9000.

Next is 'interaction.depth' which is the depth of the individual trees (number of splits in each tree). This controls the complexity of the model. The depth was tuned between 1 and 5. The learning rate is the 'shrinkage' hyperparameter and this controls the speed at which the model moves through the gradient descent. the smaller the shrinkage means the chance of overfitting is reduced, however this also increases the time to find the fit. Tuning was between 0.001 and 0.01. Lastly, 'n.minobsinnode' indicates the minimum number of observations that must be in a node in order for a split to occur which is also used to prevent overfitting, this was set to 10.

Additionally, XGBoost is variation of a boosted model and has an additional hyperparameter, 'subsample', which controls the proportion of sampled training data used in each boosting round to grow trees (without replacement). It's aim is to prevent the model from overfitting by introducing randomness. This was set to 1. Otherwise, 'XGBoost' has equivalent hyperparameters to 'GBM'.

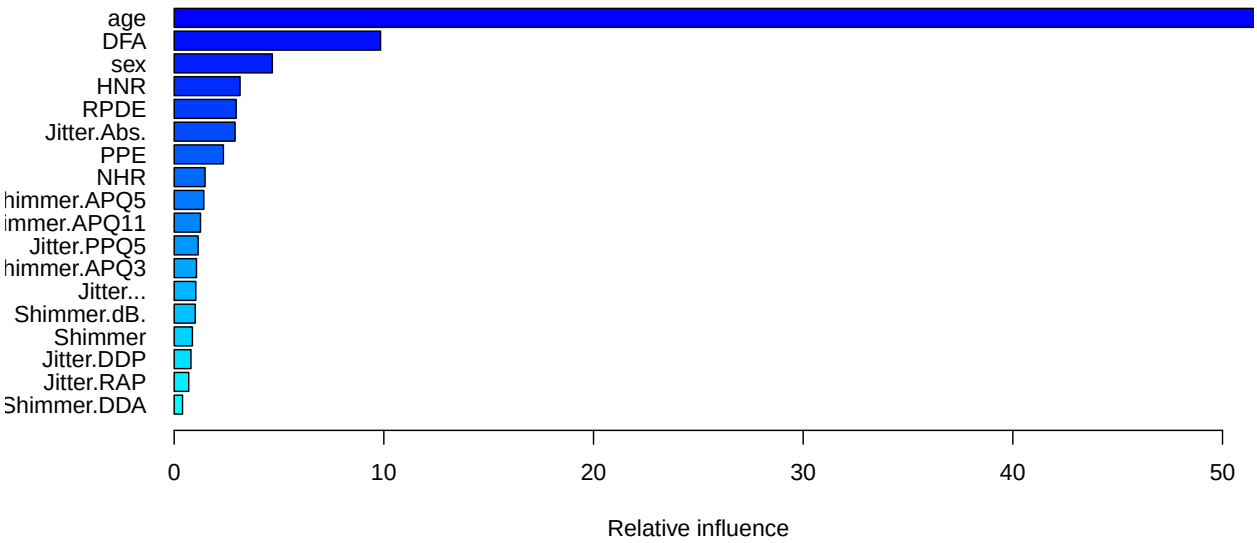

Figure 13: Variable Importance for the Gradient Boosted Model

The relative influence of the variables in the boosted model remain fairly similar to their relative influence in the Random Forest Model. This is likely due to variable importance being measured in the same way - it monitors how the loss function reduces on average from splitting each variable. This plot reaffirms age is the most important feature for predicting the response (contributes the largest reduction), followed by the DFA and sex variables.

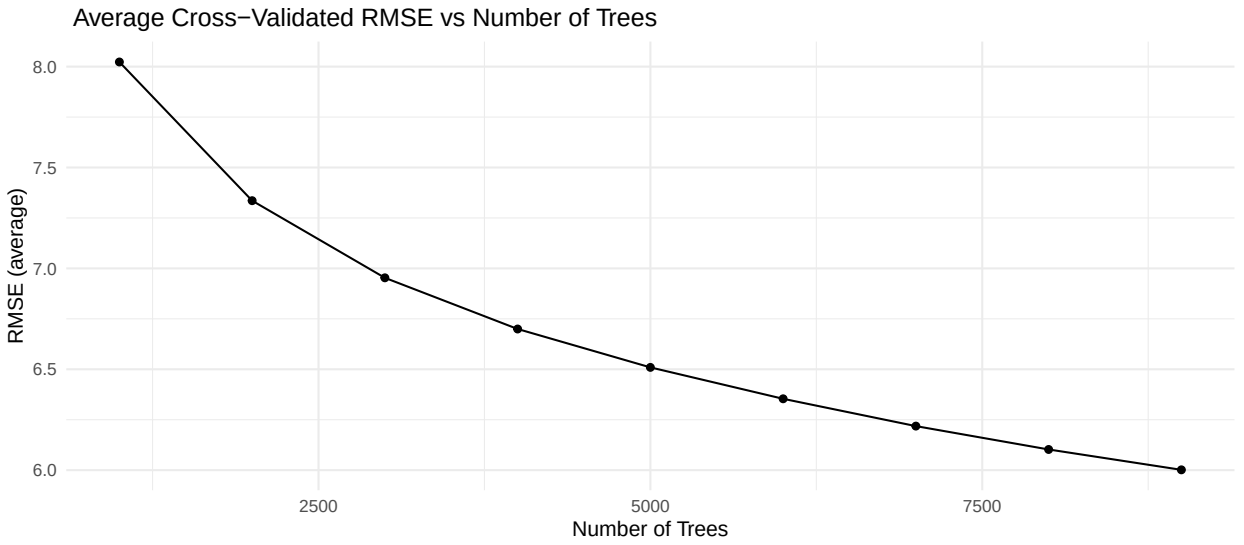

Figure 14: Graph displaying the Relationship Between the Average Cross-Validated RMSE and the Number of Trees

The figure above illustrates how increasing the number of trees in the gradient boosting model (using 'gbm'), the RMSE decreases. Thus, the response predictions increase in accuracy.

Table 10: Comparison of RMSE Across Differnt Boosted Models

GBM (6000 trees)	GBM (9000 trees)	XGB (500 trees)	XGB (750 trees)
3.8234	3.7413	3.2317	3.2414

The table above indicates (along with Figure 14), how increasing the number of trees for a 'gbm' model improves its predictive accuracy as the RMSE decreases. However, though XGBoost operates similarly, increasing the number of trees did not implicitly decrease the RMSE. This illustrates how it is Boosted Models can be difficult to tune.

Table 11: Comparison of RMSE Across Differnt Models

Linear Model with Regularisation	KNN Model	Random Forest Model	Boosted Model
10.0894	6.2832	3.1936	3.2317

The table above illustrates the RMSE values on test (out-of-sample) data for each of the best performing models within their respective model types. The linear model with regularisation predicted the response, UPDRS, with the least accuracy as it has the largest RMSE value. The KNN model performed much better than the Linear model, however not as well as the others. Based purely on the RMSE value, the Random Forest Model (RF) performed the best as it has the lowest RMSE (3.1936). However, it is important to note the boosted model's RMSE value is only marginally higher (only by 0.0381). Thus, deciding between the RF and Boosted Model to perform on unseen data is not clear cut. It is important to consider other factors.

Firstly, the RF model requires far less carefully tuned hyperparameters and take far less training time than XGBoost models. This is because RF are built in parallel, which is faster than boosted models being built sequentially. Therefore, choosing RF is more practical because it already performs well and now there is the benefit of less computational time and effort. Moreover, XGBoost is more prone to overfitting compared to RF when the models are not regularised properly and this is compounded by the fact that it is already more difficult to tune XGBoost carefully. Additionally, through RF, one can look at the relative importance of each explanatory variable across all trees, this is not as straightfoward in boosted models.

While it is important to also note that that XGBoost often out-performs Random Forest models on predictive performances and accuracy, ultimately, the Random Forest model was chosen because they easier to tune, take less computational time and are more robust to noise.

Other possible RF weaknesses are that they are not as good as XGBoost at handling both larger datasets and data with high dimensionality and the parkinson dataset is quite large. Also, if the Random Forest model was not properly tuned and has too many deep trees it could be too complex and have overfit to the training data.

Additionally, to strengthen the conviction on the choice of model and better decide which performs best, it would be beneficial to conduct further evaluations like performing error analyses and examining other error metrics.

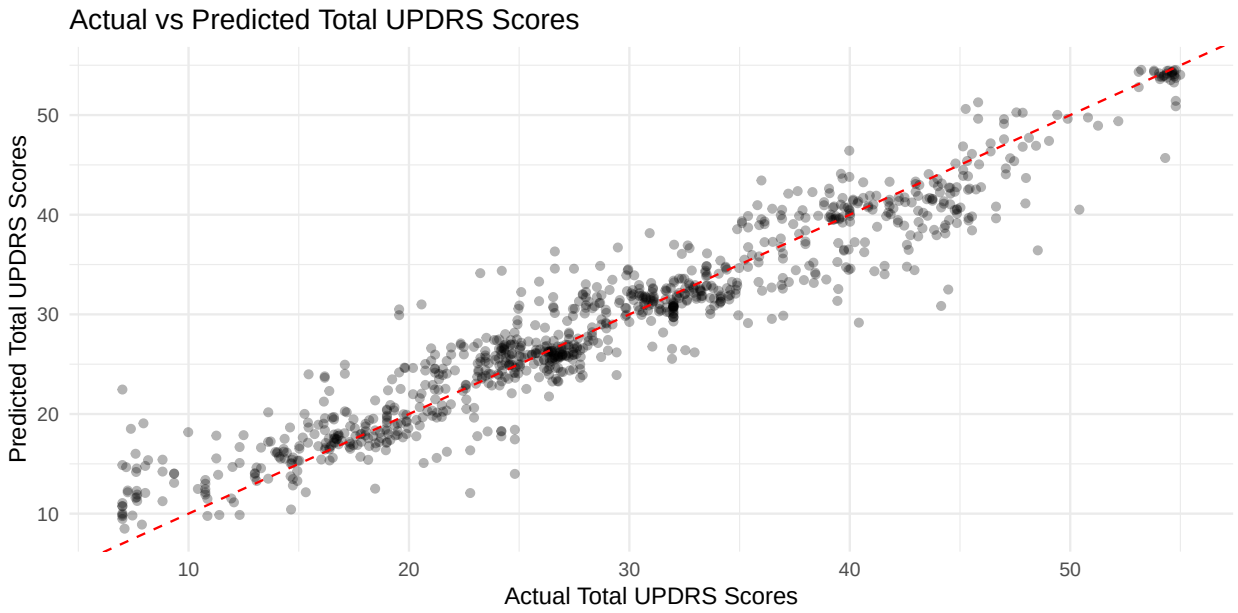

Figure 15: Random Forest Predicted Total UPDRS Compared to the Observed Total UPDRS for Out-Of-Sample Data

The figure illustrates how the points mostly lie fairly close to the diagonal line, indicating moderately accurate predictions, for along that line the actual score matches its predicted counterpart. The density of points along the line is quite strongly concentrated (seen by darker shadings), implying those points were better fit. All points above the line indicate the model over predicted those values and all below were under predicted and those values appear relatively equal in volume.

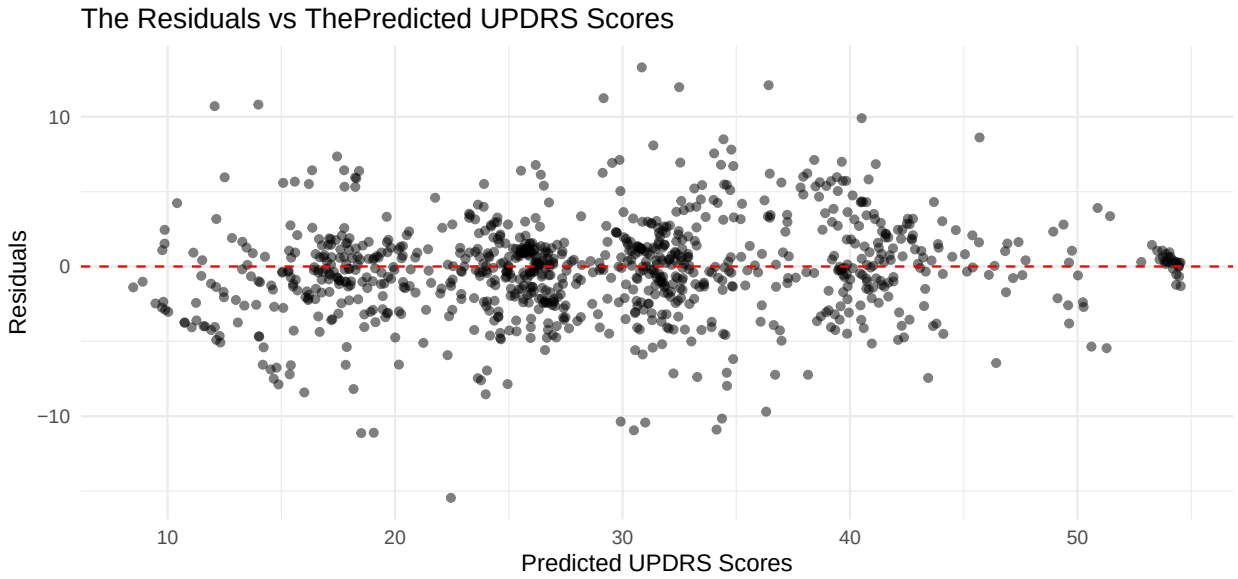

Figure 16: Relationship of residuals with the Predicted Total UPDRS for the Random Forest Model on Out-Of-Sample Data

The figure above illustrates no large fluctuations in the residuals for different values of the Predicted UPDRS score. Thus, this roughly uniform scatter indicates homoscedasticity in the model as there are no significant changes or patterns across the predicted values (randomness in the scatter). This consistent variance of the residuals suggests the model is reliable.

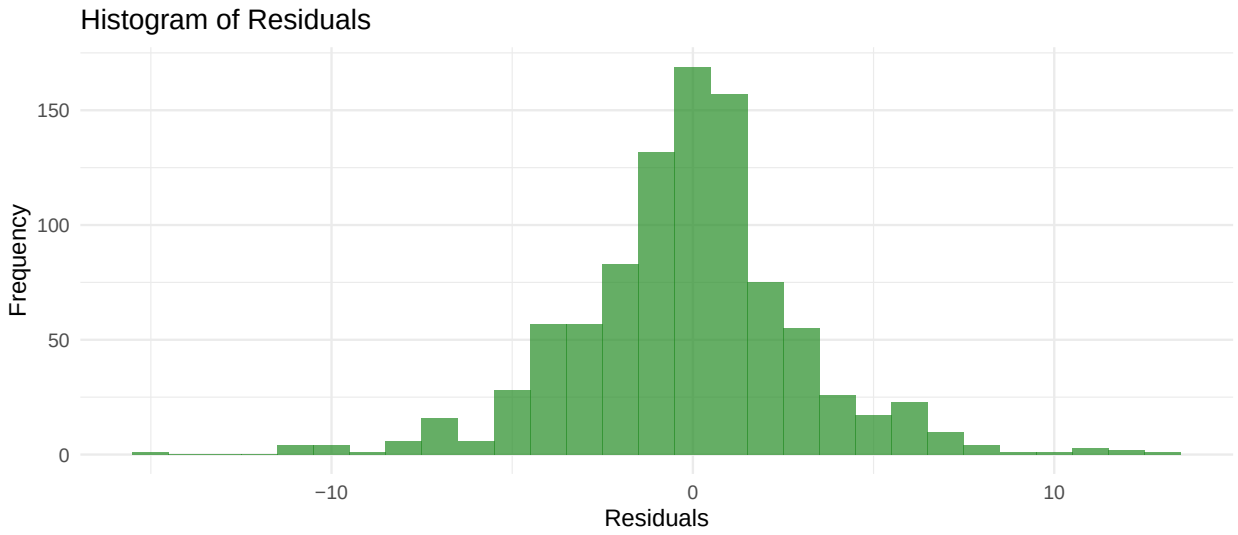

Figure 17: Histogram of the Residuals for the Random Forest Model on Out-Of-Sample Data

The residuals are centered around zero, suggesting the model does not have a significant bias and is neither systematically over or under predicting the UPDRS scores. The skewness is almost zero though slightly negative at 0.005, indicating it is mostly symmetrical. This all indicates the residuals are reasonably behaved and the model is mostly reliable.

Appendix

(1)

```
#Getting the frame from the tree
tree_frame<-vanilla_mielies$frame
#Extracting or the terminal nodes
leaves<-tree_frame[tree_frame$var=="<leaf>",]
num_term_nodes<-nrow(leaves)
#Looking at the information for each terminal node
leafinfo<-leaves$yval2
pure<-all(leafinfo[,6:9]==0 | leafinfo[,6:9]==1) #Checks the proportion of each class and if all our
                                                 #leaves only have 0's or 1's then every leaf is homogen
```

(2)

```
      ##Here we use the threshold as our decision rule and then calculate
      #the recall using the caret packages confusionMatrix() function
      pred_enet<-ifelse(pred_enet>=threshold,1,0)
      pred_enet<-as.factor(pred_enet)
      threshold<-round(threshold,3)

      caretmat<-confusionMatrix(as.factor(pred_enet),as.factor(q2_test[,1]),positive="1")
      caretmat$byClass[1]
```

```
## Sensitivity
##   0.7627119
```