An Ensemble Model to Predict Soybean Grain Yield

DA5030

Merlin Dassanayake

Summer 2024

Contents

Background Data & Motivations

Soybeans are a very important legume that is grown and used all over the world for various different products. The utility of the soybean is a huge reason as to why it is "one of the main agricultural crops in the world, with a current production estimated at 399.50 million tons". (de Oliveria et al., 2023) Crops have been selectively grown throughout history, but as we now live in an age where plant genetics give rise to genetic modification, there exist many different cultivars, or types of cultivated plants, on the market. The study by de Oliveira et al., 2023 aimed to look at 40 different cultivars and farming conditions on the overall grain yield of the crop measured in kg ha-1. Using this dataset, three different machine learning models will be used to create prediction models for the grain yield based on the various features provided. Both a homogenous and heterogenous ensemble model will also be created and each models performance and metrics will be calculated. Being able to use machine learning to help predict grain yield can provide insight into optimal growing conditions and cultivars to be used, which given the scale of soybean farming, can potentially result in a large financial and business impact for farmers.

Data Acquisition

The dataset is first loaded into R using the URL provided by the researchers. Taking a look at the structure of the data frame, we can see that there exist 320 rows of data, with 11 columns or features, which is correct.

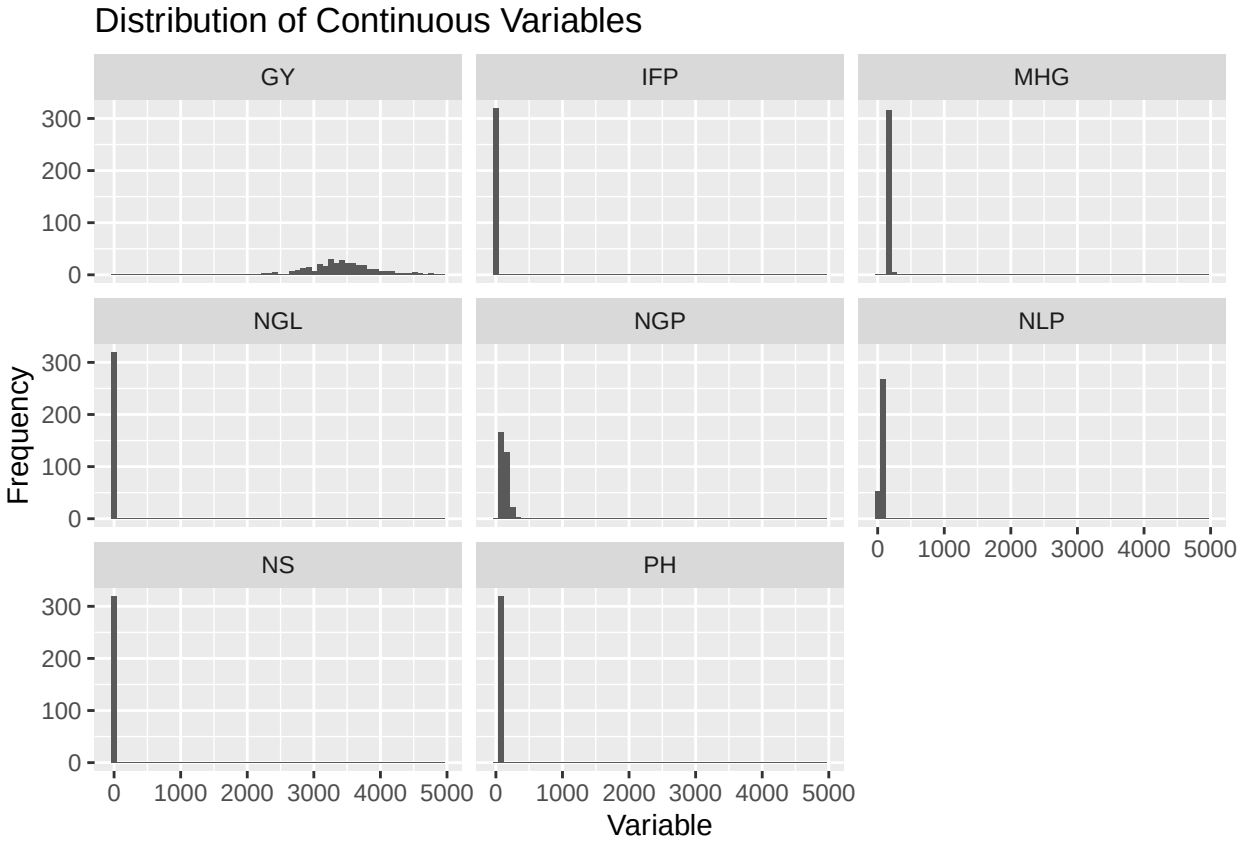
Data Exploration

```
str(soybeans)
```

```
## 'data.frame':    320 obs. of  11 variables:
##  $ Season    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Cultivar  : Factor w/ 40 levels "74K75RSF CE",..: 32 32 32 32 29 29 29 29 2 2 ...
##  $ Repetition: int  1 2 3 4 1 2 3 4 1 2 ...
##  $ PH        : num  58.8 58.6 63.4 60.3 81.2 ...
##  $ IFP       : num  15.2 13.4 17.2 15.3 18 ...
##  $ NLP       : num  98.2 102 100.4 100.2 98.8 ...
##  $ NGP       : num  178 195 203 192 173 ...
##  $ NGL       : num  1.81 1.85 2.02 1.89 1.75 1.85 1.7 1.77 2.3 2.62 ...
##  $ NS        : num  5.2 7.2 6.8 6.4 7.4 7.2 6.8 7.13 7.2 6.2 ...
```

```
##  $ MHG        : num  152 142 149 148 146 ...
##  $ GY         : num  3233 3517 3391 3313 3231 ...
```

We can then look at the distributions of all the continuous variables in the dataset.



Looking at the various distributions, we can see that only grain yield (GY) has a semblance of a somewhat normal distrubtion. This is only a concern in building our multiple regression model as it requires normally distributed data as this can improve model performance. For our regression tree and SVM models, this is not a concern so that data will not need to be transformed prior to running these models. As the GY is the one variable that has a somewhat normally distributed appearance, we can test for normality using the shapiro wilk test, in addition to the other variables.

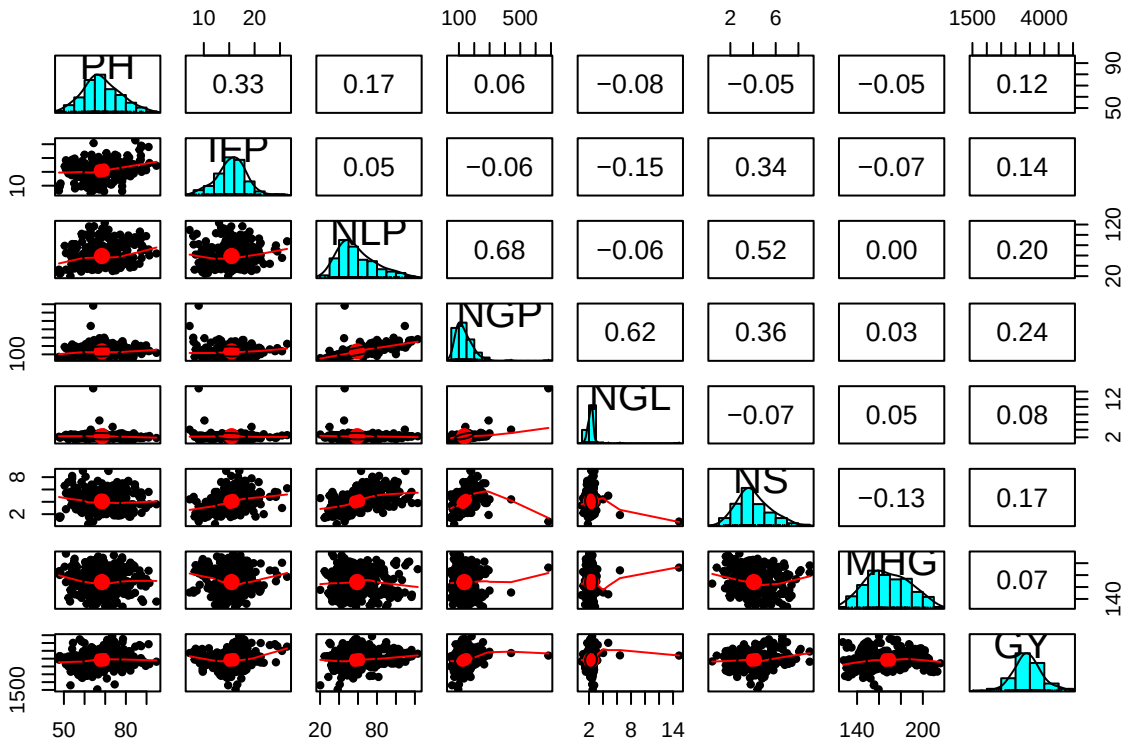
```
## [1] "The column PH is not normal."
## [1] "The column IFP is not normal."
## [1] "The column NLP is not normal."
## [1] "The column NGP is not normal."
## [1] "The column NGL is not normal."
## [1] "The column NS is not normal."
## [1] "The column MHG is not normal."
## [1] "The column GY is not normal."
```

Running a shapiro wilk test on all continuous variables returns the output that none of the distributions can be considered normal. This is concerning in the context of building our multiple regression model, however, data transformations can be conducted to mitigate this issue and improve model performance.

Next we can next look for outliers in the data frame. Using the z-score approach, the values which are 3 standard deviations or more away from the mean will be considered outliers and can be removed.

Looking at the z-scores for each continuous value, we can select for rows which contains values which are greater than 3 standard deviations. Completing this, we can see that there are 12 rows which contain outliers. These rows are now marked for removal in the data cleaning and shaping step of the process.

Next, we can look for any correlations or collinearity that might exist which could bias our multiple regression model.



Observing the graph we can see that there are no strong correlations between grain yield and the other variables. There are however, some moderate correlations between NLP, NGP, and NGL. There is a recorded .68 correlation between NLP and NGP and a recorded .62 correlation between NGP and NGL. This does make some logical sense as NLP represents the number of legumes per plant, NGP represents the number of grains per plant, and NGL represents the number of grains per pod. This is a moderate case of multicollinearity and may have an impact on the regression model which will be built using this data.

Data Cleaning & Shaping

We can then explore the data to first identify any missing values that might exist due to data recording errors.

```
# Check for missing values, returns FALSE if not NA values
any(is.na(soybeans))
```

```
## [1] FALSE
```

We can see that there do not exist any missing values, however, given the requirements of the project, I will randomly select a few values to remove and treat as missing and impute these values. Given the fairly small

number of values in the data set, removal of the values is not optimal, rather, I will impute the missing values using the mean values of the columns

```
# Verify NA values exist after removing random values
any(is.na(soybeans))
```

```
## [1] TRUE
```

```
# Impute missing values using the mean of the column
for(col in names(soybeans)) {
      soybeans[[col]] <- ifelse(is.na(soybeans[[col]]), mean(soybeans[[col]], na.rm = TRUE), soybeans[[
}

# Check for missing values after imputation
any(is.na(soybeans))
```

```
## [1] FALSE
```

After introducing random missing values, we check to see if they exist, which is verified by the first check returning TRUE. After this, I loop through each column in the data frame and test each row value to see if it is NA, if it is, it is imputed using the mean of the column ignoring the missing values. After this loop is completed, we again check to see that no NA values exist, which is verfied by the output being FALSE.

Next we will remove the outliers identified earlier. Doing so creates a data frame with 308 rows.

Now that the data frame has missing values imputed and outliers removed, we can begin to shape it for use in our three models. Some of the steps here will be the same for all models such as one hot encoding categorical variables, but other steps will be model specific and will take place in the respective section. The first change will be made to the first variable, season. Season indicates the growing season for each value, and only has two values 1 or 2. Since we will treat this as a categorical variable and one hot encoding this would result in one column, we will simply change the column to be a binary value 0 or 1. Season 1 will be changed to 1 and season 2 will be chanegd to 0. Returning the range of the Season variable after encoding shows indicates that the values are now 0 or 1.

```
##
##   0   1
## 155 153
```

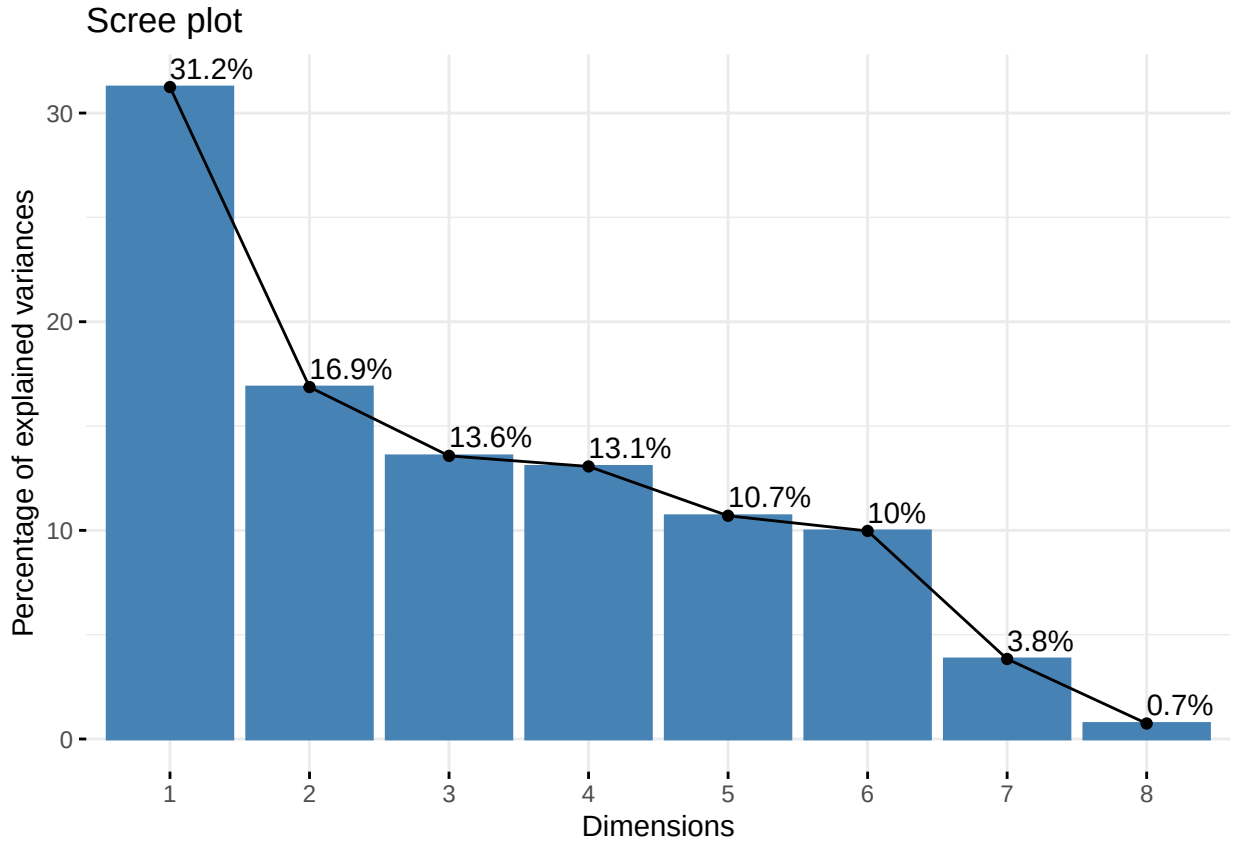
The other global change we can make is one hot encoding the repetition variable which represents the number of replications for each season, ranging from 1 to 4 repetitions. This is not a true continuous variable and will be treated as a categorical variable and will be one hot encoded.

```
##   Repetition1 Repetition2 Repetition3
## 1           1           0           0
```

We can see that this is now correctly encoded, indicating that the first row value had 1 replication for its season. As far as global changes to the data set goes, these are the changes we will make. The first shaping we will conduct is for the multiple regression model.

Building dataset for multiple regression model The multiple regression model requires further data shaping in the form of transforming variables to make then normally distributed or as close to normally distributed as possible. Then since we detected some moderate collinearity previously between NLP and NGP and NGP and NGL, we will conduct a PCA to see the amount of variance each principle component contributes to.

Here we conduct both log and sqrt transforms on the continuous variables and then conduct a shapiro wilk test to see if they are normally distrubuted after the transform. A log transform was conducted on PH and the final p-value calculated from the shapiro wilk test was 0.3944333 which is greater than 0.05 and thus the distribution can be considered normally distributed. This was conducted for all other continuous variables, and each one was transformed to a normal distribution except for the variable NGL which continued to have a very small p-value regardless of the methods of transformation to include log, sqrt, squaring, cubing, and inversion. For now we will keep this value, however, since it was one of the variables that demonstrated collinearity, further analysis needs to be done to see how much variance in the dataset NGL accounts for and if it can be removed through PCA.



Upon completion of the PCA, we can see that PH is the variable that contributes 31% of the overall variance. Components 2 to 6 all contribute in the 17% to 10 % range each, meaning IFP, NLP, NGP, NGL, and NS. Since these percentages add up to explain 100% of the overall variances, we can choose to remove features and still have at least 80% of variance explained through the remaining features. For our model, since there was moderate collinearity between NGP and NGL, and the fact that NGL was not able to be properly transformed to a normal distribution, it will be removed as a feature for the dataset used in the multiple regression model. Removing NGL from the data set still leaves 89.2% variance explained by the remaining features. At this point the data frame for our multiple regression model is ready for subsetting and usage in the model itself.

Building dataset for regression tree model At this point the data with the global changes made to it will be fine for use with our regression tree as it does not require data normalization prior to running.

Building dataset for SVM model The data set for our SVM model will need two further changes in the form of normalization and target encoding of the cultivars variable. One hot encoding of the cultivars variable does not happen automatically like the other two models and to preserve dimensionality of the data, target encoding will be performed. Normalization can be handled manually, but is also handled automatically using the ksvm function in the kernlab package so manual normalization is not necessary. One potential risk with target encoding is that using the means for each category can increase the level of overfitting in the final model as the means is derived from the target variable. To mitigate this slightly, the data for the SVM will be split prior to target encoding to prevent data leakage, or the case where testing data is used to create the model. Target encoding from the original data set without subsetting essentially will create a target mean from the training and testing data, so splitting before encoding will help to mitigate this.

```
# Verify proper target encoding
head(soybeans_svm_train[,1:3],1)
```

```
##   Season Cultivar   PH
## 1      0 3149.668 64.2
```

```
head(soybeans_svm_test[,1:3], 1)
```

```
##   Season Cultivar   PH
## 1      1 3271.418 58.8
```

After target encoding the cultivar variable we can look at the split data to see that the cultivar column now contains values of the target mean. At this point we can use the split data frames for our multiple regression model.

Model Construction

Now that our data is shaped for each model, we can split each data set into training and validation subsets for proper use with our models.

```
# Building initial model of multiple regression
multiple_regression_model <- lm(GY ~., data = soybeans_regression_train)

# Using the step function to find optimal model based on AIC
# Trace set to 0 to stop step function output from displaying
multiple_regression_model1 <- step(multiple_regression_model, trace = 0)
multiple_regression_model1$call
```

Building Multiple Regression Model

```
## lm(formula = GY ~ Season + Cultivar + PH + IFP + NGP + NS + MHG,
##     data = soybeans_regression_train)
```

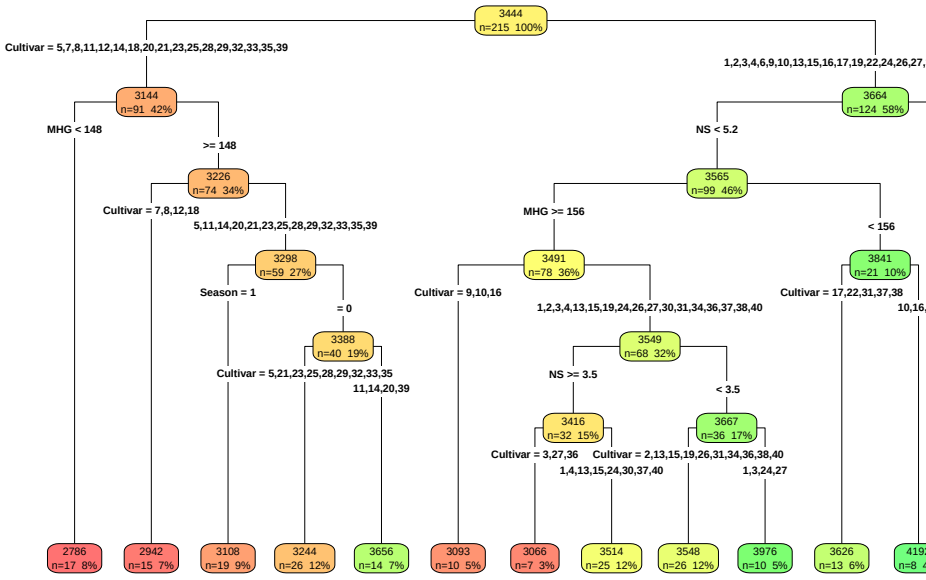
```r
# Predictions
multiple_regression_predictions <- predict(multiple_regression_model1, soybeans_regression_test)
```

Here we initially build the multiple regression model using the lm function and including every feature in the shaped and subsetted data frame. To optimize the function and remove features that will increase the performance of the model we use the step function which aims to decrease the AIC value. After this the finalized model is saved and the call for this is . The lm function automatically one hot encodes the cultivar feature so it appears as a single feature in the final call displayed above. A multiple regression model was chosen as a base model to build first as it can be used to compare the following models against. It also was chosen in the event that the data had underlying linear relationships between the target variable and its features.

```r
# Building regression tree model
rpart_model <- rpart(GY~., data = soybeans_tree_train)

# Predicting using the test variables
regression_tree_predictions <- predict(rpart_model, soybeans_tree_test, type = "vector")

# Building plot of regression tree model to visual nodes and decisions.
# Using type 4 and extra 101 to display percentages under each value
rpart.plot(rpart_model, type = 4, extra = 101, fallen.leaves = TRUE, box.palette = "RdYlGn")
```



Building Regression Tree Model

A regression tree is built using the rpart function from the rpart package. A visualization of the tree is created using the rpart.plot function in the rpart.plot package. A regression tree was chosen as the second model as it allows for the decisions to be easily understood by human viewers. In the context of the data set this is extremely important as for a farmer to increase the grain yield, physical decisions have to be made such as insertion of the first pod (IFP) as well as cultivar choice. Having these decisions be transparent allows for easy implementation of these changes in real life. According the decision tree plot, to maximize the grain yield of a soybean, a farmer should use one of the cultivars from 6,9,10,13,or 17 and they should make sure that plants have a number of stems (NS) greater than or equal to 5.2, which can be understood as greater than or equal to 5 stems per plant. These are simple changes a farmer could make that according to the model built here would improve the chance of having a grain yield of greater than 4000 kg ha-1.

```
# Building SVM model using rbfdot kernel
SVM_model <- ksvm(GY ~., data = soybeans_svm_train, kernel = "rbfdot")
# Predictions test variables
SVM_predictions <- predict(SVM_model, soybeans_svm_test)
```

Building SVM Model Finally, we build the SVM model using the rbfdot kernels which is a widely used kernel that can handle data which does not have linear relationships well. This was chosen as it is a popular kernel and one that can handle complex relationships between the data, which I suspect might be the case for this data set as it did not seem to show many correlations.

Model Evaluation

After building our models, we can then evaluate each one using metrics such as RMSE and K-fold cross validation. Here we will perform this metrics on all three models.

Calculating the RMSE for the three initial models built reveals a value of 553.59 for the RMSE for the multiple regression model, 325.96 for the regression tree, and 571.22 for the SVM model. We can see that out of the three models built, the regression tree had the lowest value of RMSE, however it is important to note that this is a singular value based of the relatively small number of values in the testing data set. We will now perform a k-fold cross validation of all three models which should give us a more robust understanding of each models performance.

```
multiple_regression_cv$results
```

```
##   intercept     RMSE Rsquared      MAE  RMSESD RsquaredSD    MAESD
## 1      TRUE 362.6459 0.398529 291.9966 58.4914  0.1739364 34.34811
```

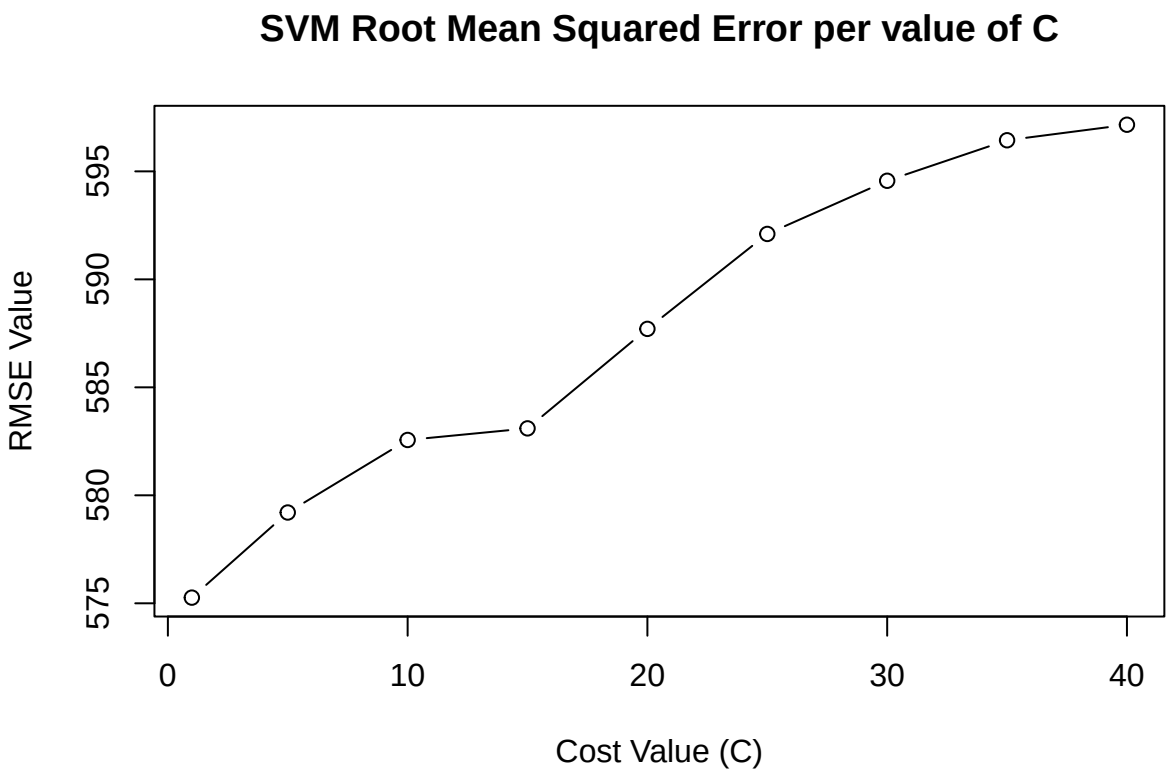
```
regression_tree_cv$results
```

```
##           cp     RMSE   Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1 0.03468020 477.3996 0.13314970 375.6009 65.62150 0.10667604 52.43985
## 2 0.07014072 467.0370 0.09586422 376.2441 52.49482 0.07148149 41.01198
## 3 0.10425678 477.2230 0.06800156 379.4201 54.74521 0.05496007 39.93361
```

```
svm_cv$results
```

```
##       sigma    C     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1 0.0587136 0.25 318.5334 0.5906646 246.4505 40.07265 0.10035030 25.62475
## 2 0.0587136 0.50 287.4087 0.6247043 223.8260 37.66897 0.09297756 24.66748
## 3 0.0587136 1.00 277.7960 0.6300521 216.4721 35.44144 0.09636182 26.50530
```

Here we can see the output of a k-fold cross validation of each model. A variety of metrics are given per model, firstly the RMSE. We can see that in this case, the regression tree seems to perform the worst while the SVM model has the lowest value of RMSE with C set to 1 at 277. As I have stated earlier, since k-fold validation takes, in this case, 10 fold of the same data with replacement and tests each model on various metrics, the output is much more robust. The SVM model also has the lowest values of mean average error (MAE). These values are important and we can see that the more complex SVM model is performing better in predicting grain yield, however there are hyperparameters such as C which can be altered to change the model performance even further.



Looking at the RMSE versus the value of the cost value (C) in the SVM model shows us a very interesting trend. It seems as though increasing the cost value in this case increases the RMSE value of the model. An explanation for this can be that as we increase the cost value, we also increase the risk of overfitting for the data, which reduces the model performance when met with novel data as in the testing data set. Based on the output of this graph we can stick with a C value of 1, which is the default value in the ksvm function and also has the lowest reported RMSE value based on this graph.

Model Tuning & Performance Improvement

In addition to these three models, ensemble models can also be used to increase the robustness of the predictions made. Here we will first create a homogeneous ensemble using bagging of our regression tree model. This model will be evaluated and then a heterogeneous ensemble model will be built out of the homogeneous ensemble, multiple regression model, and SVM model.

```
# Create homogeneous ensemble of regression tree
set.seed(1234)
# Predefining predictions vector
```

```
bagging_predictions <- vector("list",50)
# Looping through training data with replacement and saving predictions
for(i in 1:50) {
  bagging_sample <- sample(1:nrow(soybeans_tree_train), replace = TRUE)
  bagging_data <- soybeans_tree_train[bagging_sample, ]
  # Refit regression tree model using the bagging data
  bagging_model <- update(rpart_model,formula. = GY~., data= bagging_data)

  bagging_predictions[[i]] <- predict(bagging_model,soybeans_tree_test)
}
# Finding the mean prediction across 50 iterations
bagging_prediction_mean <- Reduce("+", bagging_predictions) / 50

# Calculating RMSE of the bagging model built from the regression tree
bagging_RMSE <- RMSE(soybeans_tree_test$GY, bagging_prediction_mean)
```

Here we use bagging to create a homogeneous ensemble model built on the previous regression tree model built. Decision trees are often used for bagging and in the previous evaluations, it performed poorly compared to the other two models built. We first iterate through 50 different iterations and take a sample of the training data each time with replacement. This is used to refit our original model and then predictions are made on this refitted model 50 times for the testing dataset created earlier. After doing this, the predictions are averaged across all 50 iterations to create a mean prediction for each value in the testing dataset. This dataset was then used to compare against the actual values and the RMSE value calculated was 297.74 which is lower than the original RMSE value calculated for the original regression tree model, which was 325.96. This indicates that our bagging homogeneous model has a lower level of error when predicted grain yield and should probably be used instead of the original regression tree model. Next we will use this bagging model to create a heterogeneous ensemble model using this new bagging model and our original SVM and multiple regression models.

```
# Create heterogeneous ensemble function that takes test data as input
soybean_ensemble <- function(lm_test_data, bagging_prediction_mean, svm_test_data) {
  lm_predictions <- predict(multiple_regression_model1, lm_test_data)
  svm_predictions <- predict(SVM_model, svm_test_data)
  # Returns average of each prediction
  final_predictions <- (lm_predictions + bagging_prediction_mean + svm_predictions)/3
  return(final_predictions)
}

# Running the function on the test data to output ensemble predictions
soybean_ensemble_predictions <- soybean_ensemble(soybeans_regression_test, bagging_prediction_mean, soy

# Evaluate predictions made with the heterogeneous ensemble using RMSE
soybean_ensemble_rmse <- RMSE(soybeans_tree_test$GY, soybean_ensemble_predictions)
```

Here a heterogeneous ensemble model which combines the homogeneous ensemble model created earlier, the multiple regression model, and SVM model already created. The function takes test data as input and returns the predictions of each model, and finds the mean of each prediction across the three models to return a final vector of predictions. For evaluation, the RMSE of these predictions was calculated at 460.06, which is quite a bit higher than the previous calculated RMSE for the homogeneous ensemble model. This could be do to some of the models intrinsically having a high degree of error such as the multiple regression model, which has a RMSE or 553.59. This model will obviously increase the overall RMSE for the heterogeneous ensemble model since each prediction is weighted equally. To fully optimize the heterogeneous ensemble model, weights could be applied to each prediction from each model which could update based

on the amount of error calculated in each model, which will depend on the data inputted. Overall, the heterogeneous ensemble model seems to perform worse when compared to the SVM model and homogeneous ensemble model created.

Overall, the final heterogeneous ensemble model created needs further optimization, potentially by removing the poor performing models, or by assigning weights to each prediction when calculating the mean to favor models in the ensemble which are known to perform better. As stated earlier, the models which had the lowest RMSE values were the SVM model and homogeneous ensemble model built on the regression tree model. It is not surprising that the multiple regression model struggled with this dataset as predicting grain yield based on plant statistics and farming techniques naturally would not follow a linear relationship as many processes in biology do not follow this relationship. That being said, it also makes sense that the SVM model had the lowest calculated RMSE, especially when calculated through k-fold cross validation. SVM models built with the rbfdot kernel are especially good at handling data which does not follow linear relationships, but rather more complex relationships. The only issue with this model is that it is considered a black box and it is hard to interpret for the average human. As previously touched upon, the context of this data means that understanding the decisions behind increasing grain yield are very important as these are the changes farmers can make to their crop to increase grain yield, and more importantly revenue. This is one of the reasons the regression tree model was chosen as the decisions are transparent and easily understood. The fact that the homogeneous ensemble model built through bagging upon the regression tree model performed well and had a lower RMSE than the original regression tree model makes this ensemble particularly attractive in determining grain yield.

References

de Oliveira, B. R., Zuffo, A. M., dos Santos Silva, F. C., Mezzomo, R., Barrozo, L. M., da Costa Zanatta, T. S., ... & Coelho, Y. P. (2023). Dataset: Forty soybean cultivars from subsequent harvests. Trends in Agricultural and Environmental Sciences, e230005-e230005.

UCI Machine Learning Repository https://archive.ics.uci.edu/dataset/913/forty+soybean+cultivars+from+subsequent+harvests