Introduction

Vibration signals are commonly encountered in various industrial applications, such as rotating machinery, vehicles, and aerospace systems. These signals carry important information about the health and performance of the underlying systems, which can be extracted through advanced signal processing and data analysis techniques. The goal of this report is to explore the characteristics and patterns of vibration signals in a specific system, and to develop a predictive model that can detect anomalies and potential failures of the system. The data used in this project was generated using a test rig consisting of a driving motor, two-stage planetary gearbox, two-stage parallel gearbox, and magnetic brake. Each vibration signal is recorded for a period of five minutes and with a sampling frequency of 10 kHz.

Methodology

Pre-processing

The failure data was provided in csv file containing 10,000 observations of 200 predictors. Each predictor is the size of the signal observed at a certain point in time. The response variable corresponds to a categorial class of "damage" or "undamaged" categories. Highly correlated predictors with a correlation of 0.8 were not found, nor were any near-zero variance predictors. Because of this, all 200 predictors remained in the dataset when constructing the predictive models. The code for the preprocessing is shown in Figure 1. Because we were dealing with categorial variables, it was important to determine if the observations corresponding to the two classes were balanced. This was accomplished with the code provided in Figure 2. It was determined that the datasets were balanced, so no up sampling or down sampling needed to be performed. Further preprocessing was performed in an attempt to improve model success, and this is discussed later in the statistical model section.

Statistical Model

The first model that we trained to predict the failures is a K-Nearest Neighbors model. This model was a good place to start because it can handle large amounts of predictor variables and is easy to use cross validation with. We ran a loop with 5-fold cross validation and testing each value for k from 1-10 as shown in Figure 3. This provided us with an optimal value of k = 1, and a cross validation accuracy of 80.50% as shown in Figure 4.

The next model that we constructed for this classification problem is a logistic regression model. To train this logistic regression model as a classifier, we used the train function with the generalized linear model (glm) method and no train control. To assess the effectiveness of this model, we used the divided training data with an 80/20 split. Once the model was constructed, we used it to make predictions on the holdout data and to construct a confusion matrix. The code for constructing this model is shown in Figure 5 and the confusion matrix is shown in Figure 6. As seen in Figure 6, the logistic regression model performed poorly on the validation dataset with a sensitivity of only 59.2% which is only marginally better than randomly guessing on a balanced test set.

Then, we constructed is a random forest model which uses the concept of bagging. This means that random forests are constructed with multiple deep decision trees which have low bias, and then variance needs to be combatted in construction of the model. To provide the best possibility of a successful model, we ran a loop to test out several values of the number of predictors sampled at each split of the tree, mtry in r, to determine the optimal value. We tested values of mtry between 1 and 15 because 15 is just above the square root of 200, our number of predictors, and that is the standard value of mtry used in a random forest model. During each iteration of the loop, the random forest model is constructed, and the data used is resampled to test the accuracy of the model with each value of mtry.

The best performing of these models is with a value of 14 for mtry, which makes sense as this is the closest value to the square root of our number of predictors. During the testing of the values of mtry, we held the number of trees at a constant value of 200. Before testing values of mtry, we tested a few different values for the number of trees. We settled on 200 trees because that is around the point where the marginal increase in accuracy with having more trees began to be outweighed by the computational cost and time of training the model. The final model in which mtry equals 14 provides an accuracy of 73.05% which is an improvement on the logistic regression model, but still not as accurate as the benchmark we are trying to beat. The code and model summary for the random forest model are shown in Figure 7 and Figure 8, respectively.

After using bagging with a random forest model, we used boosting. Specifically, we used a gradient boosting machine to construct a new classification model. With gradient boosting, we are taking advantage of the low variance of small trees and then combatting bias when training the model. Because we were not testing different values of other parameters in this model, we decided on a final value of 1000 trees in order to increase the potential accuracy of the model. Just as with the KNN model, we used 5-fold cross validation in the training of this model. The cross-validation of this model is only 58.94%, putting it on par with the unsuccessful logistic regression model. The code and cross-validation results are shown in Figure 9 and Figure 10.

After not seeing the desired results from construction of the previously discussed classification models, we decided to revisit preprocessing the data. Although it didn't make sense to remove any predictors, we decided that it still may be beneficial to reduce the number of predictors to simplify the work needed to construct the models. We achieved this by combining multiple predictors into one. This reduces the number of predictors without necessarily losing much information. The new data frame is a result of taking the average of a certain number of predictors throughout the data frame to create a new, smaller set of predictors. This way, the patterns of the signal data are still mostly left intact while the number of predictors is reduced. We tried this method 3 times; grouping the variables in groups of 5, then 4, then 2. These sets of modified predictors were all first tested using the KNN model because that was the most successful model up to this point. The best performance out of these 3 sets was when the data were reduced to 100 variables, each the average of 2 of the originals. The code for the creation of these new variables is shown in Figure 11, and the KNN model using this new dataset is shown in Figure 12. Once again, cross validation is used in training this KNN model. The optimal value of K remains at K = 1 for this model. As seen in Figure 13, the accuracy of this model is slightly lower than that of the original KNN model at a value of 79.97%. With the tests of the groups of 4 and 5 predictors being reduced to 1, the accuracy of the KNN model decreases more with a smaller number of predictors. Because of time, KNN was the only model tested with a reduced number of predictors. To further improve the model in the future, we would apply the reduced predictor sets to the other model types.

After testing all of the models constructed so far, we did not pass the final benchmark, so we decided to create a model that combined the strengths of a few of the models we created that had performed reasonably well. To do this, we created a new data frame the consisted of three columns. Each column represented predictions on the test data from a different model. Then, a fourth column is created that contains the mode of the first three columns. This means that this fourth column contains the majority guess from the three models and uses this as its prediction, so that at least two of the three models would have to agree to classify as either damaged or undamaged for the prediction to be made. This allows for the strengths of the different models to be combined to improve accuracy. Originally, we tested the original KNN model, the random forest model, and the gradient boosting model together. These three models were originally selected because they had the best performance on the validation data and were all different types of models. However, this model performed worse than the KNN model

by itself. This is likely because the gradient boosting model performed only marginally better than randomly guessing, so it actually hurt this model. Because this model failed to improve accuracy, we used the KNN model with a reduced number of predictors in place of the gradient boosting model. This was the final model we used, and it outperformed all our other models when tested against the Kaggle leaderboard and it surpassed all the benchmarks set. The construction of this model is shown in Figure 14.

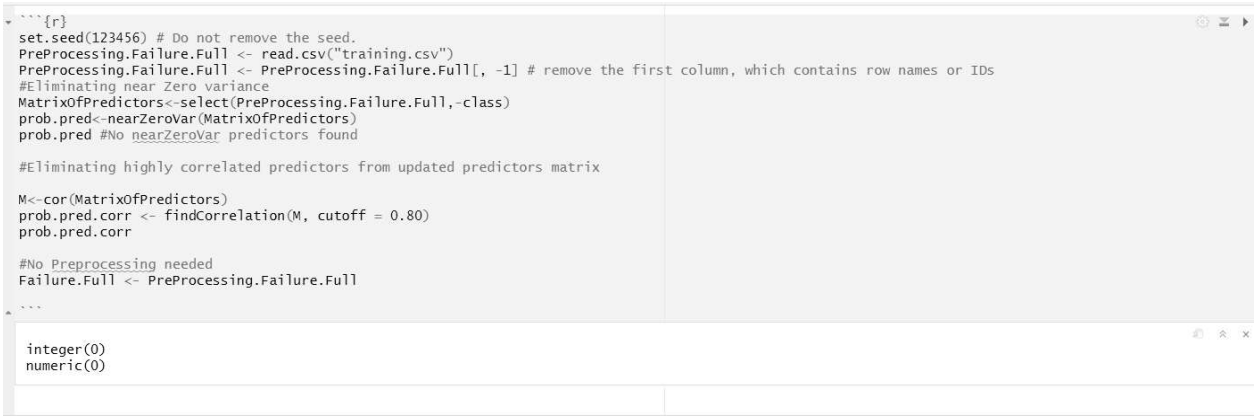
Results

In the public leaderboard, our final model performed just above the highest benchmark of 0.83888 with an accuracy of 0.84444. However, when the private leaderboard was published, the accuracy of our model dropped a little bit to 0.82583, but the accuracy of the highest benchmark dropped as well to 0.82000. This may seem to be a result of overfitting to the test data. However, all of the validation that we performed for our model was done with a subset of the training data or with cross validation. Because of this, the reduction in accuracy is likely just a result of variability in the specific observations selected for the public and private leaderboards. Overall, our model was mostly successful in its objective of identifying failures in the parts. This can be partially seen in a relatively high accuracy. However, a true measure of this would be in the sensitivity in the performance of the model which would provide the ratio of failed parts that were identified as failed. However, this data is not what is measured by the leaderboards, so the success of the model in terms of catching failures cannot be accurately measured. Because the models composing our final model used cross validation in training, a confusion matrix was not created, so there is not an easy way to look at the sensitivity of our final model.

Conclusions

There were several models tested throughout the analysis of this data and we saw varying levels of success from the different model types. Out of the original testing of the model types learned in class, the K-Nearest Neighbors model with k = 1 performed the best, followed by the random forests model. The KNN model likely outperformed the other models due to the large number of predictors included in the model. Because each variable is an individual measurement of the signal observed, information would be lost by removal of predictors. Random forest likely performed well because of the depth of the trees and low bias which works well with a higher number of predictors compared to gradient boosting. The most successful model was a combination of the best performing models. This approach succeeded because it helped reduce the effects of the weaknesses of each type of model. Overall, we succeeded in trying a variety of approaches to determine how to best predict failures on the test data. To further improve the model, we could do more research on classification methods to try out. We could also try to apply the adjusted predictor set to the other model types to see if there would be an improvement with the models that work better with less predictors, such as logistic regression. Since the main goal of the problem is to accurately detect problems (high sensitivity) rather than just a high accuracy, our final model could be modified to only predict the undamaged class if all three models agreed that it was undamaged. This would hurt the specificity but improve the sensitivity for the model.

Appendix



```r
set.seed(123456) # Do not remove the seed.
PreProcessing.Failure.Full <- read.csv("training.csv")
PreProcessing.Failure.Full <- PreProcessing.Failure.Full[, -1] # remove the first column, which contains row names or IDs
#Eliminating near Zero variance
MatrixOfPredictors<-select(PreProcessing.Failure.Full,-class)
prob.pred<-nearZeroVar(MatrixOfPredictors)
prob.pred #No nearZeroVar predictors found

#Eliminating highly correlated predictors from updated predictors matrix

M<-cor(MatrixOfPredictors)
prob.pred.corr <- findCorrelation(M, cutoff = 0.80)
prob.pred.corr

#No Preprocessing needed
Failure.Full <- PreProcessing.Failure.Full
```

```
integer(0)
numeric(0)
```

Figure 1 – R Code for Preprocessing

```
#Test if data is balalanced

counts <- table(Failure.Train$class)
barplot(counts, main="Bar chart for whole data", ylab = "Frequency",
   xlab="Class",  col = "lightblue",
   cex.axis = 1.0, cex.lab = 1.3, cex.main = 1.2)
grid(nx = NULL, ny = NULL)

#It is balanced
```

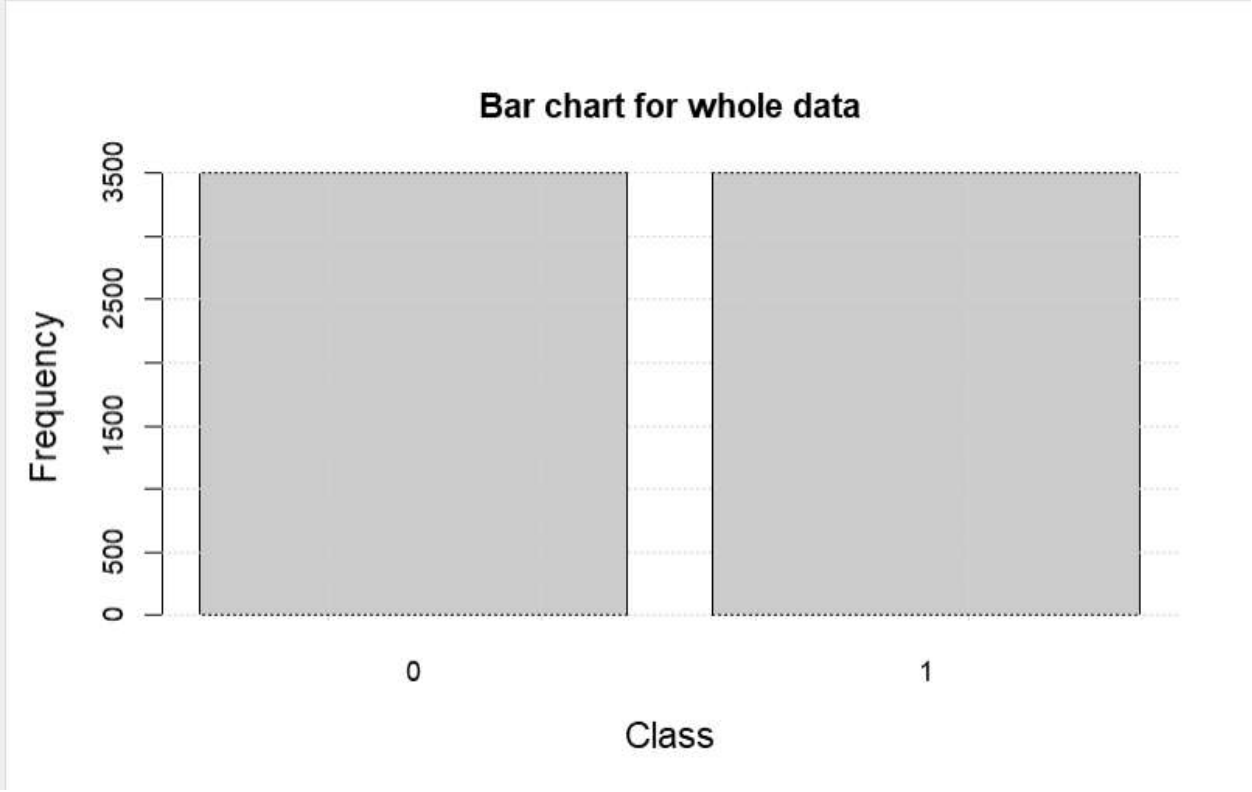


Figure 2 – R Code to test if the data in unbalanced

```
```{r, fig.align='center'}
Failure.Train <- mutate(Failure.Train, Class.Train)

train_control <- trainControl(method = "cv", number = 5)
# Fit a model.
set.seed(4323687)
KNNfit <- train(class~.,
                data = Failure.Full, method = 'knn',
                preProc = c("center", "scale"),
                trControl = train_control,
                tuneGrid = expand.grid(k = seq(1, 10, by = 1)))
print(KNNfit)
plot(KNNfit)

#Determined that k = 1 is optimal

#Generating Predictions for the Test Data
KNN.Predict <- predict(KNNfit, newdata = Test.Data, type = 'prob')
```

Figure 3- K-Nearest Neighbors Model Training
```

```
k-Nearest Neighbors

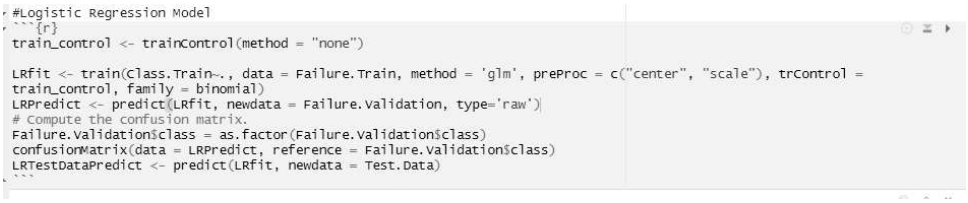
10000 samples
  200 predictor
    2 classes: '0', '1'

Pre-processing: centered (200), scaled (200)
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 8000, 8000, 8000, 8000, 8000
Resampling results across tuning parameters:

  k   Accuracy  Kappa
   1  0.8050    0.6100
   2  0.7570    0.5140
   3  0.7734    0.5468
   4  0.7510    0.5020
   5  0.7493    0.4986
   6  0.7321    0.4642
   7  0.7319    0.4638
   8  0.7244    0.4488
   9  0.7142    0.4284
  10  0.7046    0.4092

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 1.
```

Figure 4- K-Nearest Neighbors Parameter Testing Summary



```
#Logistic Regression Model
```{r}
train_control <- trainControl(method = "none")

LRfit <- train(Class.Train~., data = Failure.Train, method = 'glm', preProc = c("center", "scale"), trControl =
train_control, family = binomial)
LRPredict <- predict(LRfit, newdata = Failure.Validation, type='raw')
# Compute the confusion matrix.
Failure.Validation$class = as.factor(Failure.Validation$class)
confusionMatrix(data = LRPredict, reference = Failure.Validation$class)
LRTestDataPredict <- predict(LRfit, newdata = Test.Data)
```
```

Figure 5- R Code for Construction and Validation of Logistic Regression Model

```
Confusion Matrix and Statistics

          Reference
Prediction  Damaged Undamaged
  Damaged       592       458
  Undamaged     408       542

               Accuracy : 0.567
                 95% CI : (0.5449, 0.5889)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.123e-09

                  Kappa : 0.134

 Mcnemar's Test P-Value : 0.0959

            Sensitivity : 0.5920
            Specificity : 0.5420
         Pos Pred Value : 0.5638
         Neg Pred Value : 0.5705
             Prevalence : 0.5000
         Detection Rate : 0.2960
   Detection Prevalence : 0.5250
      Balanced Accuracy : 0.5670

       'Positive' Class : Damaged
```

Figure 6- Confusion Matrix for Logistic Regression Model

```r
```{r}

predictors.train <- Failure.Train
Failure.Full$class <- as.factor(Failure.Full$class)
Failure.Full$class <- ifelse(Failure.Full$class == 0, "Undamaged", "Damaged")

oob_train_control <- trainControl(method="oob",
                                  classProbs = TRUE,
                                  savePredictions = TRUE)
Failure.Train <- mutate(Failure.Train, Class.Train)
my.m <- floor(sqrt(6))
rf.tune.grid <- expand.grid("mtry" = seq(1,15))
forestfit.OOB <- train(class ~ .,
                 data = Failure.Full, method = 'rf',
                 trControl = oob_train_control,
                 tuneGrid = rf.tune.grid,
                 ntree = 200, # Number of trees
                 nodesize = 1, # Minimum terminal node size.
                 cutoff = c(0.5, 0.5)) # Cutoff to for two classes in the trees.
# It takes slightly longer than a standard decision tree.
print(forestfit.OOB)



Forest.Predict <- predict(forestfit.OOB, newdata = Test.Data)
Forest.Predict <- ifelse(Forest.Predict == "Undamaged", 0,1)
```

Figure 7- R Code for Construction Random Forest Model
```

```
Random Forest

10000 samples
  200 predictor
    2 classes: 'Damaged', 'Undamaged'

No pre-processing
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
   1    0.7013    0.4026
   2    0.7206    0.4412
   3    0.7171    0.4342
   4    0.7191    0.4382
   5    0.7297    0.4594
   6    0.7224    0.4448
   7    0.7260    0.4520
   8    0.7223    0.4446
   9    0.7231    0.4462
  10    0.7275    0.4550
  11    0.7242    0.4484
  12    0.7257    0.4514
  13    0.7299    0.4598
  14    0.7305    0.4610
  15    0.7257    0.4514

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 14.
```

Figure 8- Accuracy Table for each Value of mtry in the Random Forest Models

```r
```{r}
set.seed(1000)
Failure.Train <- mutate(Failure.Train, Class.Train)
Failure.Train$Class.Train <- factor(Failure.Train$Class.Train)
Failure.Validation$class <- factor(Failure.Validation$class)
Failure.Full$class <- factor(Failure.Full$class)
my.params <- expand.grid("n.trees" = 1000, "shrinkage" = 0.01,
                         "interaction.depth" = 1,
                         "n.minobsinnode" = 10)
train_control <- trainControl(method="cv", number = 5)

gbm.fit <- train(class~ .,
                 data = Failure.Full, method = 'gbm',
                trControl = train_control,
                tuneGrid = my.params,
                verbose = FALSE) # Suppress part of the output from the function.
# It takes slightly longer than a standard decision tree.


gbm.predict = predict(gbm.fit, newdata = Test.Data)
print(gbm.fit)

```
```

Figure 9- R Code for Construction and Validation of Gradient Boosting Model

```
Stochastic Gradient Boosting

10000 samples
  200 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 8000, 8000, 8000, 8000, 8000
Resampling results:

  Accuracy  Kappa
  0.5894    0.1788

Tuning parameter 'n.trees' was held constant at a value of 1000
Tuning parameter 'interaction.depth' was
 held constant at a value of 1
Tuning parameter 'shrinkage' was held constant at a value of 0.01
Tuning
 parameter 'n.minobsinnode' was held constant at a value of 10
```

Figure 10- Model Summary for Gradient Boosting Model

```
#Averaging 2 Columns together to have a smaller number of predictors
```{r}
Failure.Train2 <- read.csv("training.csv")
Failure.Train2.Class <- Failure.Train2$class
Failure.Train2.Class <- as.factor(Failure.Train2$class)
Failure.Train2 <- select(Failure.Train2, -id, -class)
# create 40 new variables, each representing the average of 5 of the original variables
for (i in 1:100) {
  start_col <- (i-1)*2 + 1
  end_col <- i*2
  new_var_name <- paste0("avg_", i)
  Failure.Train2 <- Failure.Train2 %>% mutate(!!new_var_name := rowMeans(.[start_col:end_col]))
  Test.Data <- Test.Data %>% mutate(!!new_var_name := rowMeans(.[start_col:end_col]))


}
avg_Failure.Train2 <- select(Failure.Train2, starts_with("avg_"))
avg_test.data <- select(Test.Data, starts_with("avg_"))
```
```

Figure 11- R Code for Reducing the 200 Predictors into 100 that are Each an Average of 2

```
```{r}
Failure.Train <- mutate(avg_Failure.Train2, Failure.Train2.Class)

train_control <- trainControl(method = "cv", number = 5)
# Fit a model.
set.seed(4323687)
KNNfit2 <- train(Failure.Train2.Class~.,
                 data = Failure.Train, method = 'knn',
                 preProc = c("center", "scale"),
                 trControl = train_control, •
                 tuneGrid = expand.grid(k = seq(1, 5, by = 1)))
print(KNNfit2)
plot(KNNfit2)

KNN.Predict2 <- predict(KNNfit2, newdata = avg_test.data, type='prob')
names(KNN.Predict2) <- c("Undamaged", "Damaged")
```
```

Figure 12- Construction of KNN Model with Reduced Dataset

```
k-Nearest Neighbors

10000 samples
  100 predictor
    2 classes: '0', '1'

Pre-processing: centered (100), scaled (100)
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 8000, 8000, 8000, 8000, 8000
Resampling results across tuning parameters:

  k  Accuracy  Kappa
  1  0.7997    0.5994
  2  0.7488    0.4976
  3  0.7771    0.5542
  4  0.7586    0.5172
  5  0.7718    0.5436

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 1.
```

Figure 13- Results of K-Fold Cross Validation for KNN Model with Reduced Dataset

```
#Combined Model
```{r}
names(KNN.Predict) <- c("Undamaged", "Damaged")

pred_df <- data.frame(pred1 = Forest.Predict, pred2 = KNN.Predict$Damaged, pred3 = KNN.Predict2$Damaged)
pred_df <- pred_df %>%
  rowwise() %>%
  mutate(mode_col = names(sort(table(c(pred1,pred2,pred3)), decreasing = TRUE))[1])

print(pred_df)
output_col <- select(pred_df, mode_col)
write.csv(output_col, "TestX2.csv")
```
```

A tibble: 3,000 × 4 Rowwise:

pred1 <dbl>	pred2 <dbl>	pred3 <dbl>	mode_col <chr>
1	1	1	1
1	1	1	1
1	0	0	0
1	1	1	1
0	0	0	0
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1
0	0	0	0

Figure 14- R Code and Data Frame of Final Model Combining 3 Models

Statements of Contribution:

Joshua Jowers- I organized meeting times and divided the work for creation of models and writing of the report. I also lead the effort for development of the KNN models, and the combining of variable s and combining models for our final model. I wrote most of the descriptions of the models in the report.

Bennett Foret – I wrote the introduction and conclusion for the report and provided help with the creation of the models. I provided help at all team meetings and attended all meetings.

Enmanuel De La Gala – I contributed to the redaction of sections of this report like preprocessing and statistical model, as I also contributed to the development of different models (like the KNN, gradient boosting and tree classifications) that were helpful in the development of our final model.