

# TORONTO BUS DELAYS

ST3082 Project II  
Advanced Data Analysis



Group 04

- • • s14853 – Pramudi Rajamanthri
- • • s15030 – Vidura Chathuranga
- • • s15091 – Chalani Wijamunige

## **Abstract**

This report aims to outline the findings of the exploratory data analysis which was carried out based on the Toronto bus delays 2022 data set obtained from the open-source data site, Kaggle. Furthermore, an advanced analysis was carried out on identifying the frontline reasons for Toronto bus delays along with their inter relationships and fitting a suitable model to predict the delay time of buses with high accuracy.

## **Contents**

Abstract.....	1
List of figures.....	1
List of Tables.....	1
1. Introduction.....	1
2. Description of the Question.....	2
3. Description of the data set.....	2
4. Pre-processing.....	2
4.1 Feature Engineering.....	2
5. Important Results of the Descriptive Analysis.....	3
6. Important Results of the Advanced Analysis.....	4
7. Issues Encountered and Proposed Solutions.....	7
8. Discussion and Conclusions.....	7
9. Appendix of the code.....	8

## **List of Figures**

Figure 5.1 : Summary plot.....	3
Figure 5.2 : Distribution of Gap(In minutes).....	3
Figure 5.3 : PLSR - VIP scores .....	4
Figure 5.4 : Silhouette plot .....	4
Figure 5.5 : Cluster plot.....	4
Figure 6.1 : Variable importance plot on Random Forest.....	5
Figure 6.2 : Variable importance plot on XGBoost.....	5
Figure 6.3 : Log delays distribution.....	6
Figure 6.4 : Scatterplot of Min.Gap Vs log delays.....	6
Figure 6.5 : Variable importance plot on Random Forest(Log transformed response).....	7
Figure 6.6 : Variable importance plot on XGBoost(Log transformed response).....	7
Figure 8.1 : Partial Dependence plot.....	7

## **List of Tables**

Table 3.1 : Variable Summary.....	2
Table 4.1 : Route Description.....	2
Table 6.1 : Model performance evaluation on the original data.....	4
Table 6.2 : Results of Random Forest and XGBoost models.....	5
Table 6.3 : Model performance evaluation on the log transformed response.....	6
Table 6.4 : Model performance evaluation on the log transformed response.....	6

### **1. Introduction**

In Toronto, buses play a major role in public transportation. As per records in 2021, TTC bus system has 192 bus routes in operation including 28 night bus routes carrying over 264 million riders over 6686 kilometers of routes with buses travelling 143 million kilometers in the year. The system had a ridership of about 8,74,300 per weekday by the end of first 6 months of 2022. Bus routes extend throughout the city and are integrated with the subway system and the streetcar system with free transfers among the 3 systems. In the recent past years, TTC is constantly being complained on the delays of the bus system despite of having a wide network. Numerous research studies have been done on the factors leading to these delays in order to fix this issue with the guidance of the professionals, since the primary objective of transportation efficiency is directly affected from these delays in the bus system. With the dataset in hand, an analysis can be performed to find the factors that have the most influence on bus delays in Toronto and a model can be built in order to predict the delay time in minutes accurately for the ease and the awareness of the riders.

## **2. Description of the Question**

It is important and timely to address the issue in drop of efficiency in the Toronto bus system due to delays, in order to fix the problematic spots in the bus network, as it is directly connected with the day today life activities of millions of people. Hence, identifying and understanding the main factors resulting this issue is essential in setting the path to predict the delay time in minutes out of the visible causes such as route types, incidents causing delays, week of the day etc, to make the passenger lives easy by being aware of the possible amount of delay times.

## **3. Description of the data set**

The Toronto bus delays 2022 dataset contains 27351 records and 10 variables, including Delay time as the response variable.

Variable Name	Description	Variable Type
Date	Date of the delay incident	Qualitative
Route	Route number which the incident happened	Qualitative
Time	Time of the day which the incident happened	Quantitative
Day	Day of the week which the incident happened	Qualitative
Location	Location in Toronto which the delay incident happened	Qualitative
Incident	The incident which caused the delay	Qualitative
Min.Delay	Amount of time which the bus got delayed	Quantitative
Min.Gap	Time gap in minutes with the next bus scheduled	Quantitative
Direction	Direction of the route which the particular bus travels	Qualitative
Vehicle	Vehicle number of the bus	Qualitative

*Table 3.1*

## **4. Pre-processing**

Toronto bus delay dataset contains data under 10 variables where 7 are categorical and 3 are numerical. In the dataset, 154 duplicated entries were discovered and those were removed. Then it was found that there are 164 missing data records in the *Route* variable and 5529 missing records in the “*Direction*” variable. Afterwards, all the records having 0 as the *Min.Delay* and *Min.Gap* were removed from the dataset. By doing so, the count of missing values of *Route* and *Direction* variables got reduced to 26 and 4987 respectively with a total of 25898 records.

### **4.1 Feature Engineering**

A separate “Month” variable was created by extracting the months in the “Date” variable. Distinct values in the “*Direction*” column were identified and entries with error values discovered in that variable were filtered out. Considering the “*Route*” variable, routes labelled in text form were replaced by a random number for the ease of cleaning process.

1.Regular and Limited Service Routes	7 – 189 series	Weekdays – 6am-1am Weekends - 8am-1am
2.Blue Night Routes	300 -399 series	Weekdays – 1am-6am Weekends - 1am-8am
3.Community Routes	400 – 499 series	Only in rush hours of weekdays Rush hours (According to TTC statistics) :- Morning – 6am- 10am Evening - 3pm- 7pm
4.Express Routes	900 – 999 series	Any time with in the 24 hours (according to requirements)

*Table 4.1*

Based on the above details we got from the TTC official website, the new variable *Route\_New* was created by categorizing all the distinct routes in the *Route* variable into 5 categories (Blue Night Routes, Community Routes, Express Routes, Regular and limited service routes, Others) based on their route numbers. The *Hour* variable was introduced to the dataset by extracting the hours in the *Time* variable. The new variable *is\_Weekday* was formed by categorizing the days in the *Day* column into 2 categories (Weekday, Weekend). The *Incident\_New* variable was created by taking the distinct categories in the *Incident* column and combining 3 categories with lowest number of cases (Cleaning – Disinfection, Held By, Late Entering Service)

into one category (Others), and renaming the category “Road Blocked – NON-TTC Collision” into “Road Blocked”. The *Vehicle* variable was removed from the dataset as it acts as an identifying variable and it causes no effect for the delay analysis.

Furthermore, by referring to the website of TTC bus system, the operating days and times of the bus routes were identified and the records which do not follow this day and time system under its corresponding bus route were filtered out as they are contradictory with the system data which left us with a total of 23853 records. Finally, the dataset was split into training and test data sets as 80% and 20% from the original data respectively for further analysis. Afterwards the variables with missing values of the training set were taken into consideration and we identified missing values in *Route*, *Route\_New* and *Direction* variables as 24, 24, 3528 respectively. As *Route* variable is not considered for further analysis while *Direction* variable leads to misinterpretations with imputing using the mode as all its categories have approximately equal number of data currently, we only imputed the *Route\_New* variable using the *Hour* and the *is\_Weekday* observations relevant to those records with data in Table 4.1.

## 5. Important Results of the Descriptive Analysis

In the detailed descriptive analysis carried out previously, the behavior of the delay time was analyzed with several associative factors. Below are some important results obtained.

- When the delay time distribution was analyzed, a positively skewed distribution was observed along with 994 outliers. Hence, considering a transformation on the response would help to overcome the issues arising from the above.
- Most delay cases were reported in the regular and limited service routes( 7-189) as it functions from 6am-1am and 8am-1am on weekdays and weekends respectively .Furthermore the buses which travel towards North Toronto tend to report more delays.

• Weekdays record the most number of delay cases with in 6am-7pm particularly as those are the identified rush hours in weekdays according to the statistics of TTC. Mechanical problems and delays in operations due to the operators are the major reasons which were identified as the causes of the delays with in these identified time period.

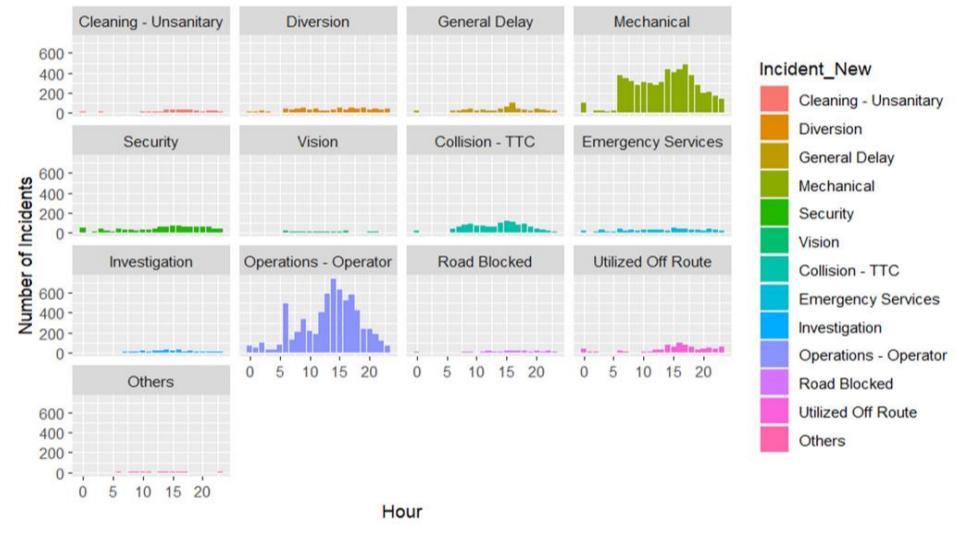


Figure 5.1

- The time gap between the scheduled time of two buses shares a positive linear relationship with the delay time, along with a considerably higher correlation of 0.96.

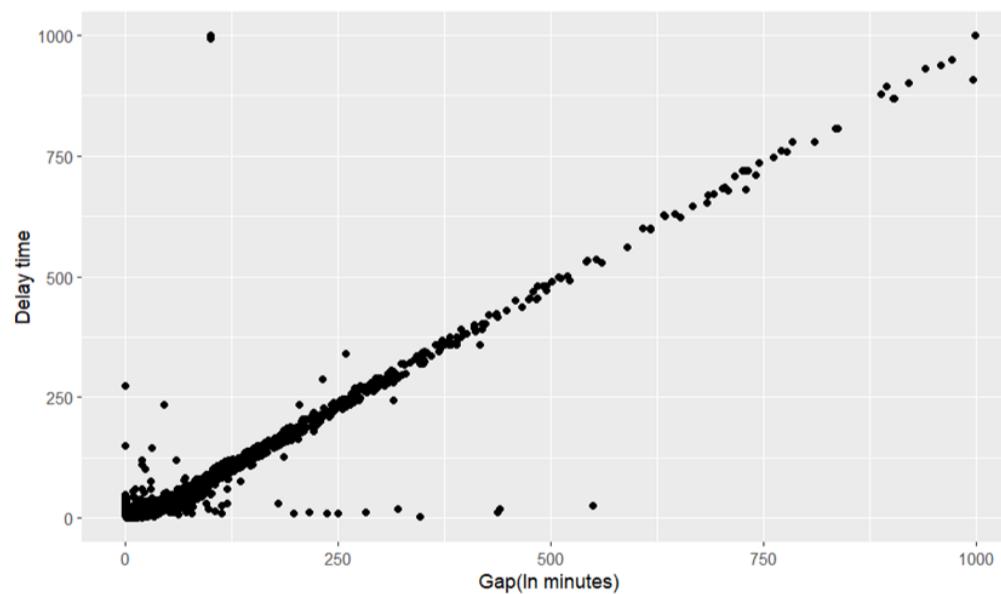


Figure 5.2

- Partial least squares regression(PLSR),also resulted a dominant VIP score for Min.Gap which confirmed the identified linear relationship between Min.Gap and Min.Delays . Moreover, 35 outliers were identified via PLSR.

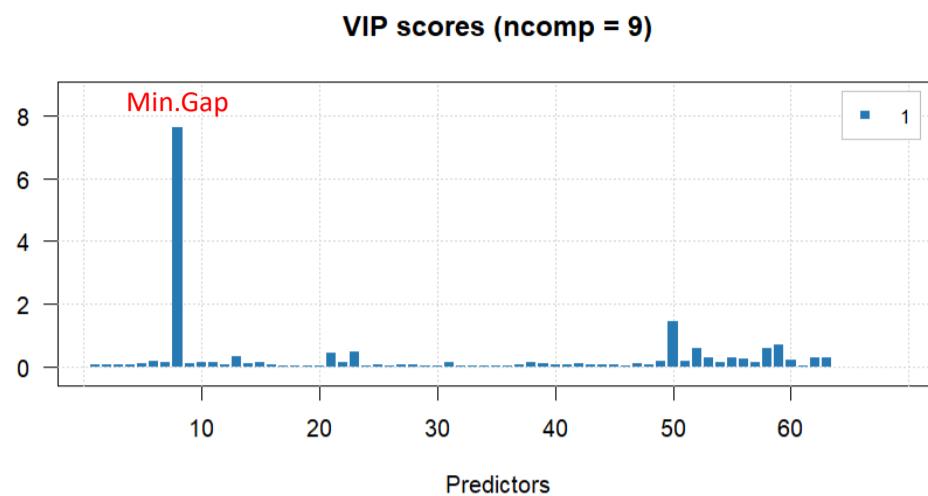


Figure 5.3

- Cluster analysis confirmed the non-existence of clearly distinct clusters where it resulted average silhouette distances less than 0.1 for the considered k values in the range of 2:8 where the highest value was 0.0894 under k=3.

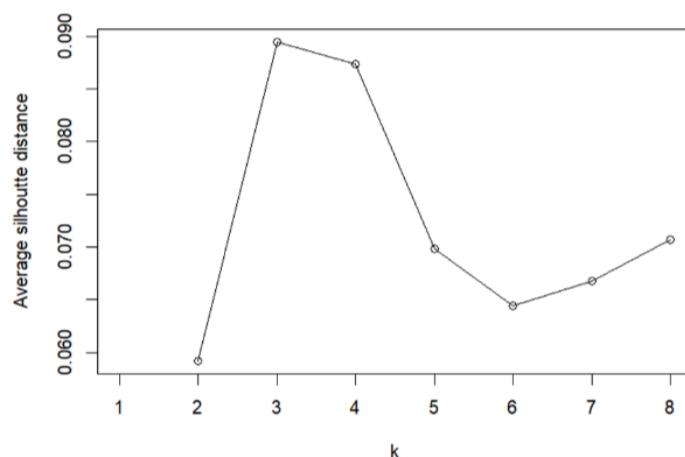


Figure 5.4

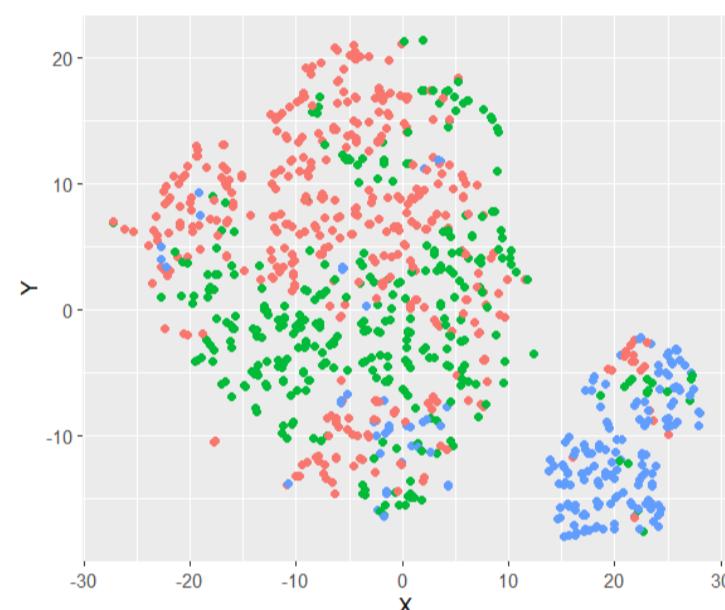


Figure 5.5

- Apart from the strong linear correlation of 0.96 identified between *Min.Gap* and *Min.Delay*, strong associations were identified among the *Min.Gap* and categorical predictors such as *Route\_New* , *Incident\_New* and *Hour* etc through the Kruskal Wallis results where there were strong associations among the categorical predictors identified through chi squared test results also.

## 6. Important Results of the Advanced Analysis

The predictive models analyzed throughout this analysis are derived with the assumption that the *Min.Delay* variable is of continuous data. Accordingly, with regard to the final result of the descriptive analysis mentioned above, fitting a multiple linear regression model is not appropriate due to the presence of associations identified among the predictor variables. Hence, regularization techniques were used to reduce the variance of the coefficient estimates. Table 6.1 below is a model performance evaluation done on the predicted delays and that of actual counts using the Root Mean Squared Error (RMSE) metric with 10- fold cross validation.

Model	Training RMSE	Test RMSE	Test MAPE
Ridge	4.919888	7.255654	15.89%
Lasso	4.93747	7.271554	16.16%
Elastic Net	4.797923	7.509318	22.75%

Table 6.1

Comparing the table values of the Test RMSE for the three models, clearly the ridge model reports the lowest, furthermore it has the lowest difference between the training and test RMSE values depicting to have the maximum minimization of the issue of overfitting. But, obtaining 15.89% MAPE value suggests a considerably less prediction accuracy relevant to the ridge model.

Hence, selecting a comprehensive model by considering the above three models might not be fair enough.

Hence, to improve the model accuracy regarding the MAPE, more sophisticated but that can handle multiple issues simultaneously were applied.

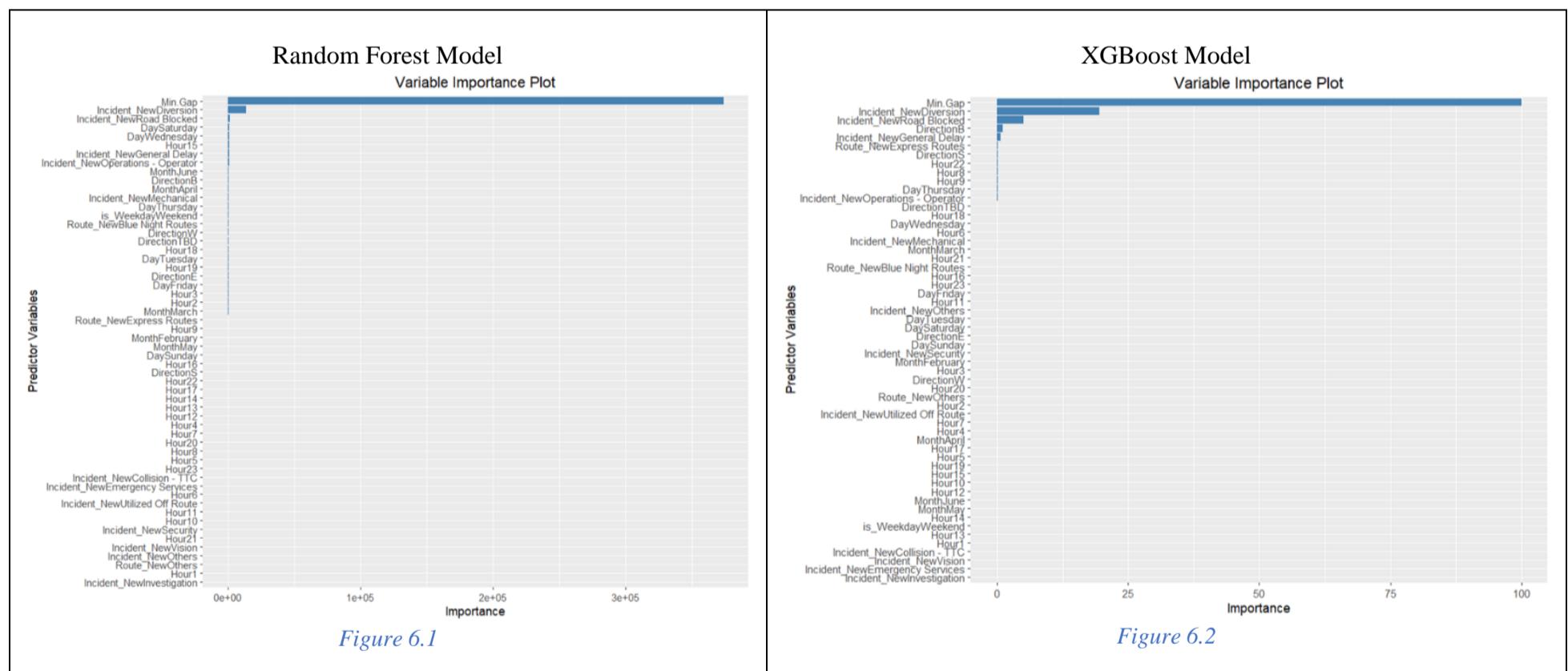
Table 6.2 below are the results obtained using random forest and XGBoosting after hyper parameter tuning.

Model	With all the features				With the important features		
	Best Parameters	Training RMSE	Test RMSE	Test MAPE	Training RMSE	Test RMSE	Test MAPE
Random Forest	mtry=50 splitrule=variance min.node.size=5	3.595297	5.278197	9.18%	3.746567	6.524895	14.75%
XGBoost	nrounds=500 max_depth=2 eta=0.1 gamma=0.1 colsample_bytree=0.6 min_child_weight=1 subsample=0.5	2.799788	5.520246	9.57%	3.868539	6.530456	14.86%

Table 6.2

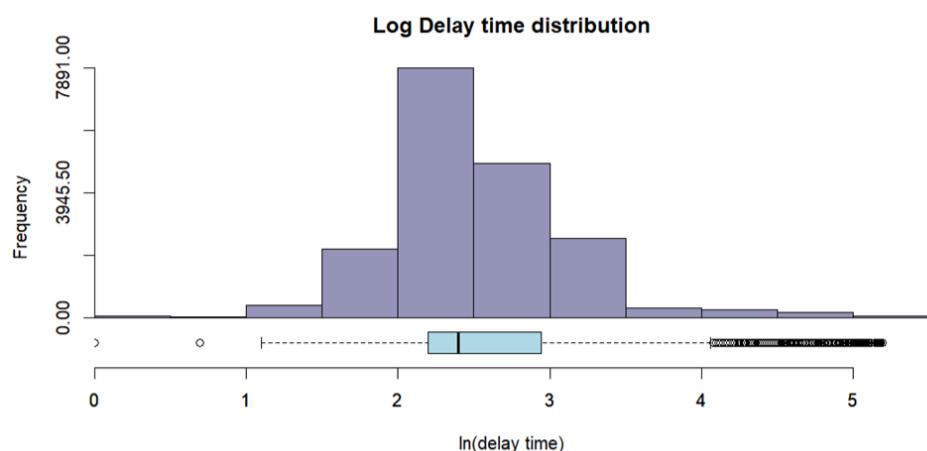
Note: the important features for the Random Forest and XGBoosting were decided by taking all the features that have scores greater than 100 and 0.1 respectively.

As depicted on Table 6.2, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy has increased but comparatively the random forest model with all the features shows better results with both less Test RMSE and Test MAPE. Despite all that, Test MAPE values seem to have reduced compared to all the regression models applied in this case. Moreover, the random forest and xgboost models considering all the features reported training MAPE values of 7.56% and 5.58% respectively, approving the fact that random forest model with all the features is less overfitted. Even though, models with the important features were fitted, the prediction accuracy has decreased with considerably higher Test MAPE values. Table below shows the variable importance plots of the two models with all the features considered.



From both models we observe that the variable *Min.Gap* has resulted a dominant importance in predicting the delays, leaving majority of other variables negligible. This observation coincides with the result that we identified in the descriptive analysis where the scatterplot of *Min.Gap Vs Min.Delay* depicted a perfect positive linear relationship with a correlation coefficient of 0.96.

The response variable's discreteness and the possession of a positively skewed distribution indicates that using a transformation on the response might produce more accurate predictions. Therefore, log transformation is imposed on the response, *Min.Delay*.



	Response – Min.Delay	Response – Log(Min.Delay)
Skewness	5.192685	0.6769573
Kurtosis	37.93428	5.319729

Figure 6.3

Table 6.3 below is a model performance evaluation done on the predicted delays after transforming them back to the original scale and that of actual counts using the Root Mean Squared Error (RMSE) metric with 10-fold cross validation.

Model	Training RMSE	Test RMSE	Test MAPE
Ridge	30.85677	42.79989	35.59%
Lasso	25.99747	38.51643	25.42%
Elastic Net	33.03114	48.24348	35.75%

Table 6.3

Above results show that the log transformation has not been successful in fitting linear relationships between the response and predictors where the figure 6.4 confirms that the perfect positive linear relationship between Min.Delay and Min.Gap has become non-linear between ln(Min.Delay) and Min.Gap as it is identified as the most important predictor in this scenario. Hence we decided to shift to the random forest and xgboost methods from the regularization techniques in search of more accuracy.

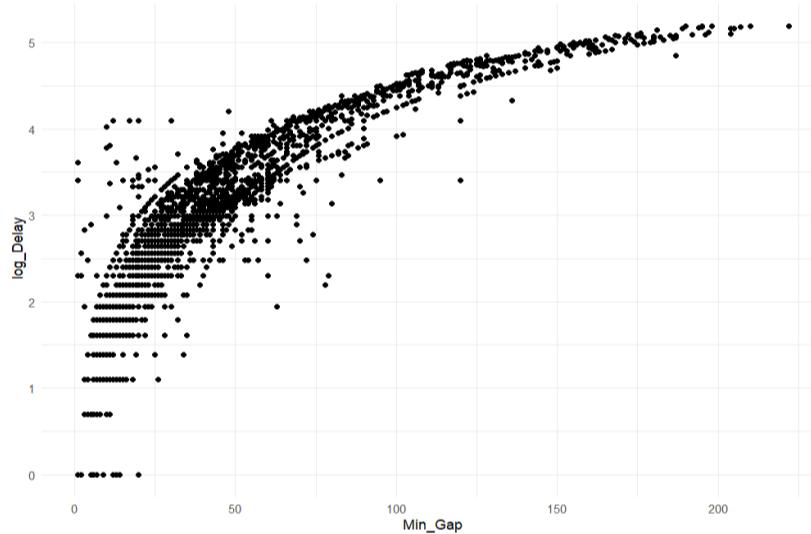


Figure 6.4

Table 6.4 below are the results obtained using random forest and XGBoosting after transforming them back to the original scale.

Model	With all the features				With the important features		
	Best Parameters	Training RMSE	Test RMSE	Test MAPE	Training RMSE	Test RMSE	Test MAPE
Random Forest	mtry=35 splitrule=variance min.node.size=5	4.571086	6.377511	13.49%	2.654157	6.973025	14.82%
XGBoost	nrounds=500 max_depth=2 eta=0.1 gamma=0.1 colsample_bytree=0.5 min_child_weight=1 subsample=0.6	3.172071	6.110895	11.97%	3.450382	6.312643	13.28%

Table 6.4

Note: the important features for the Random Forest and XGBoosting were decided by taking all the features that have scores greater than 0.1.

As depicted on Table 6.4, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy is high in terms of both RMSEs and Test MAPE but lower comparatively to the results we obtained for the random forest model and XGBoost models considering the response in original scale. Even though, models with the important features have increased the accuracy in the training data, the prediction accuracy has decreased related to the test data with a considerable amount of overfitting and higher Test MAPE values. Figure 6.5 and Figure 6.6 depict the variable importance plots relevant to this scenario with all features considered.



## 7. Issues Encountered and Proposed Solutions

- Presence of a large number of factor levels in a categorical variable would make the model very complicated. Therefore, before fitting the models some factor levels with very low representation were combined together with the intention of reducing the number of factor levels in a categorical variable
- The dataset which was used for the analysis comprised of categorical variables with many factor levels. In order to fit a model, these variables had to be replaced with dummy variables. As a result, the different factors were separated into many dummy variables, it caused difficulties in interpreting the model since each factor level for the same categorical variable had different levels of importance where some factors had extremely high feature importance while the factors had very low feature importance.
- When almost all the models didn't show vast differences in terms of training and test RMSEs, in order to identify which models are overfitted, we considered their training and test MAPEs, particularly for the models with considerably less training and test RMSEs which paved the path to select the best model.

## 8. Discussion and Conclusions

- Even though our initial distribution of the response variable was highly skewed to the right, the models which were fitted under the regularization techniques Ridge, lasso and Elastic Net resulted fair enough training and test RMSE values.
- It was evident that *Min.Gap* variable is dominant among all the variables in all the variable importance plots coinciding with the fact it shares a correlation of 0.96 with the response. Due to the above reason the results of the models with all the features and only the important features didn't result vast differences in terms of RMSE values.
- The results obtained in random forest model on the response in original scale considering all the features showed the highest accuracy regarding the test RMSE and test MAPE values which are 5.278197 ,9.18% respectively. Moreover, the obtained training MAPE was reported to be 7.56% which approves a lower overfitting of the model comparing with the test MAPE. The obtained results after log transformation relevant to the ensemble techniques also were fair enough but not better than the results obtained from the random forest model on the response variable.

- When all the models were compared, the Random Forest model applied on the original response considering all the features was identified as the best model which produces the most accurate predictions with less information loss. Figure 8.1 shows the partial dependency plot of Min.Gap which is the only numerical variable in the selected random forest model where it depicts the predicted delay time increases as the time gap increases in the model.

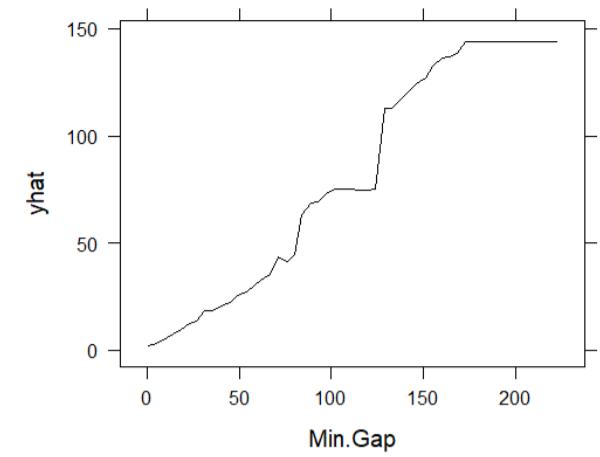


Figure 8.1

## 9. Appendix of the code

```

1  Data=read.csv("D:/3RD YEAR/SEMESTER II/ST3082/Project I Materials//ttc-bus-delay-data-2022.csv")
2  View(data)
3  head(data)
4  summary(data)
5  str(data)
6  library(plyr)
7  library(dplyr)
8  library(stringr)
9  library(mgsub)
10 library(ggplot2)
11 library(corrplot)
12 library(psych)
13 library(packHV)
14 library(moments)
15 library(tidyverse)
16 library(Metrics)
17
18
19 ##### Pre- processing and Feature Engineering #####
20 #Removing duplicates
21 sum(duplicated(data))
22 data=distinct(data)
23 nrow(data)
24
25 #Replacing "" with NA
26 data = replace(data, data=="", NA)
27 colSums(is.na(data))
28
29 #Removing records with delay=0
30 data=data[data$Min.Delay != 0, ]
31 nrow(data)
32
33 #Removing records with gap=0
34 data=data[data$Min.Gap != 0, ]
35 nrow(data)
36
37
38 #Create Month column
39 data = data %>%
40   mutate(Month = str_extract(Date, "[alpha]+"))
41 ) %>%
42
43   mutate(
44     Month = ifelse( str_detect(Date, "Jan"), "January", Month)
45   ) %>%
46
47   mutate(
48     Month = ifelse( str_detect(Date, "Feb"), "February", Month)
49   ) %>%
50
51   mutate(
52     Month = ifelse( str_detect(Date, "Mar"), "March", Month)
53   ) %>%
54
55   mutate(
56     Month = ifelse( str_detect(Date, "Apr"), "April", Month)
57   ) %>%
58
59   mutate(
60     Month = ifelse( str_detect(Date, "May"), "May", Month)
61   ) %>%
62
63
64 #Cleaning Direction column
65 table(data$Direction)
66 which(data$Direction=="/")
67 which(data$Direction=="2")
68 which(data$Direction=="6")
69 which(data$Direction=="0")
70 which(data$Direction=="J")
71 which(data$Direction=="")
72 which(data$Direction=="9")
73 data=data %>% filter(!row_number() %in% c(76,19133,21555,6669,11507,152,4249,10326,13986))
74 nrow(data)
75
76 #Create Route_New column
77 x=c(table(data$Route))
78 which(data$Route=="RAD")
79 which(data$Route=="OTC")
80 data$Route[5619]=1000
81 data$Route[7346]=1000
82 data$Route[19679]=1000
83 data$Route[13033]=1001
84
85 Route_New<-()
86 for(i in 1:nrow(data)){
87   if((is.na(data$Route[i]))&& (as.numeric(substr(data$Route[i],1,4))>=7)&(as.numeric(substr(data$Route[i],5,6))<=1000)) {
88     Route_New[i]="Regular and Limited service routes"
89   } else if((is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>299)&(as.numeric(substr(data$Route[i],5,6))<=299)) {
90     Route_New[i]="Blue Night Routes"
91   } else if((is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>399)&(as.numeric(substr(data$Route[i],5,6))<=399)) {
92     Route_New[i]="Community Routes"
93   } else if((is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>899)&(as.numeric(substr(data$Route[i],5,6))<=899)) {
94     Route_New[i]="Express Routes"
95   } else if(is.na(data$Route[i])){
96     Route_New[i]="Others"
97   } else{
98     Route_New[i]=NA
99   }
100 }
101 table(Route_New)
102 data=cbind(data,Route_New)
103
104
105
106
107
108
109
110 #Create Hour column
111 Hour=c()
112 for(i in 1:length(data$Time)){
113   Hour[i]=as.numeric(substr(data$Time[i],1,regexpr(":",data$Time[i])-1))
114 }
115 data=cbind(data,Hour)
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224

```



```

593 #For the test set
594 y_hat_enet = predict(rf_model, data2[,-2])
595 actual=y2
596 pred=c(y_hat_enet )
597 mape_m = mape(y2, pred)
598 rmse_m = rmse(y2, pred)
599 mape_m
600 rmse_m
601
602 # get variable importance
603 # Print the variable importance measures
604 var_imp=rf_model$finalModel$variable.importance
605 # print the variable importance
606 print(var_imp)
607
608 # load ggplot2 library
609 library(ggplot2)
610
611
612 var_imp = data.frame(Variables = names(rf_model$finalModel$variable.importance),
613                       Importance = rf_model$finalModel$variable.importance,
614                       row.names = NULL)
615
616 ggplot(data = var_imp, aes(x = reorder(Variables, Importance), y = Importance)) +
617   geom_bar(stat = "identity", fill = "steelblue") +
618   labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
619   theme(plot.title = element_text(hjust = 0.5)) +
620   coord_flip()
621
622 #####
623 # For the test set
624 df=var_imp[order(-var_imp$Importance),]
625 View(df)
626 df[c(1:41),]
627 nrow(df)
628
629 # Calculate number of variables to be removed
630 vars_to_remove = df$Variables[42:56]
631
632 # Remove least important variables from data1
633
634 dummy_data1= dummyVars(~ . , data = data1,sep = "")
635 data1_encoded= predict(dummy_data1, newdata = data1)
636
637 View(data1_encoded)
638 data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
639 View(data1_imp)
640 colnames(data1_imp)
641
642 base_var=c("Route_NewRegular and limited service routes","DayMonday","MonthJanuary","Incident_NewCleanin")
643 data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
644 View(data1_imp)
645 str(data1_imp)
646 data1_imp=as.data.frame(data1_imp)
647 # Convert dummy variables to factors
648 num_cols = c("Min.Delay", "Min.Gap")
649 # Convert non-numeric columns to factors
650 factor_cols = setdiff(names(data1_imp), num_cols)
651 data1_imp$factor_cols = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))
652
653 # Check the data types
654 str(data1_imp)
655 set.seed(100)
656 indexes=sample(1:nrow(data1_imp),0.1*nrow(data1_imp))
657 data_new=data1_imp[indexes,]
658
659
660 # Define the tuning grid
661 tune_grid = expand.grid(
662   mtry = 41,
663   splitrule = "variance",
664   min.node.size = c(1, 5, 10,15)
665 )
666
667 # Tune the hyperparameters using the tune() function
668 library(ranger)
669
670
671 # Perform 5-fold cross-validation
672 set.seed(100)
673 rf_model = train(
674   Min.Delay ~ .,
675   data = data_new,
676
677   method = "ranger",
678   trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
679   tuneGrid=tune_grid,
680   importance="impurity"
681 )
682
683 best_params = rf_model$bestTune
684 plot(rf_model)
685 #For the training set
686 y_hat_enet = predict(rf_model, data1_imp[,-7])
687 actual=y1
688 pred=c(y_hat_enet)
689 rmse_m = rmse(y1, pred)
690 rmse_m
691
692
693 #####
694 #For the test set
695 # Calculate number of variables to be removed
696 # Remove least important variables from data1
697
698 dummy_data2= dummyVars(~ . , data = data2,sep = "")
699 data2_encoded= predict(dummy_data2, newdata = data2)
700
701 View(data2_encoded)
702 data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
703 View(data2_imp)
704 colnames(data2_imp)
705 data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
706 View(data2_imp)
707
708 data2_imp=as.data.frame(data2_imp)
709
710 # Convert non-numeric columns to factors
711 factor_cols = setdiff(names(data2_imp), num_cols)
712 data2_imp$factor_cols = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))
713
714 y_hat_enet = predict(rf_model, data2_imp[,-7])
715
716 actual=y2
717 pred=c(y_hat_enet )
718 rmse_m = rmse(y2,pred)
719 rmse_m
720 mape_m = mape(y2, pred)
721 mape_m
722 mape_m
723
724
725
726
727
728 #Xgboost
729 library(xgboost)
730 Xtrain = xgb.DMatrix(data = as.matrix(data1[,-2]),label=y1)
731 y_train = data1$Min.Delay
732 Xtest = xgb.DMatrix(data = as.matrix(data2[,-2]),label=y2)
733 y_test = data2$Min.Delay
734
735 #####
736
737 set.seed(100)
738 xgb_model = train(Xtrain,
739   y_train,
740   method = "xgbTree",
741   objective = "reg:squarederror",
742   trControl = trainControl(method = "cv",
743                           number = 5,
744                           verboseIter = TRUE),
745
746
747   tuneGrid = expand.grid(nrounds = c(500,1000),
748                         eta=0.1,
749                         max_depth = c(2,4,6),
750                         colsample_bytree = c(0.5,0.6),
751                         subsample = c(0.5,0.6),
752                         gamma=0.1,
753                         min_child_weight = 1
754
755   ))
756
757 plot(xgb_model)
758 y_hat_enet = predict(xgb_model, Xtrain)
759 actual=y1
760 pred=c(y_hat_enet)
761 rmse_m = rmse(y1, pred)
762 rmse_m
763
764 #For the test set
765
766
767 y_hat_enet = predict(xgb_model, Xtest)
768
769 actual=y2
770 pred=c(y_hat_enet )
771 mape_m = mape(y2, pred)
772 rmse_m = rmse(y2, pred)
773 mape_m
774 rmse_m
775
776
777 # get variable importance
778 var_imp=varImp(xgb_model)
779
780 # print the variable importance
781 print(var_imp)
782
783 # load ggplot2 library
784 library(ggplot2)
785
786
787 # plot the variable importance
788 ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
789   geom_bar(stat = "identity", fill = "steelblue") +
790   labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
791   theme(plot.title = element_text(hjust = 0.5))
792
793
794 #####
795 # Calculate number of variables to be removed
796
797 vars_to_remove = df$Variables[23:56]
798
799 # Remove least important variables from data1
800
801 dummy_data1= dummyVars(~ . , data = data1,sep = "")
802 data1_encoded= predict(dummy_data1, newdata = data1)
803
804
805 xgb_model = train(Xtrain ~ .,
806                     y_train,
807                     method = "xgbTree",
808                     objective = "reg:squarederror",
809                     trControl = trainControl(method = "cv",
810                                   number = 5,
811                                   verboseIter = TRUE),
812
813                     tuneGrid = expand.grid(nrounds = c(500,1000),
814                               eta=0.1,
815                               max_depth = c(2,4,6),
816                               colsample_bytree = c(0.5,0.6),
817                               subsample = c(0.5,0.6),
818                               gamma=0.1,
819                               min_child_weight = 1
820
821
822 plot(xgb_model)
823 y_hat_enet = predict(xgb_model, Xtrain)
824 actual=y1
825 pred=c(y_hat_enet)
826 rmse_m = rmse(y1,pred)
827 rmse_m
828 rmse_m= sqrt(sum((actual -pred)^2)/nrow(data1))
829
830 #For the test set
831
832 dummy_data2= dummyVars(~ . , data = data2,sep = "")
833 data2_encoded= predict(dummy_data2, newdata = data2)
834
835
836 View(data2_encoded)
837 data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
838 View(data2_imp)
839
840 colnames(data2_imp)
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999

```

```

871 str(data2_im)
872 data2_im<-as.data.frame(data2_im)
873 # Convert dummy variables to factors
874 num_cols = c("Min_Delay", "Min.Gap")
875 # Convert non-numeric columns to factors
876 factor_cols = setdiff(names(data2_im), num_cols)
877 data2_im[factor_cols] = apply(data2_im[factor_cols], 2, function(x) as.factor(x))
878 Xtest = xgb.DMatrix(data = data2_im[, -4], label=y2)
879 y_test = data2$Min.Delay
880
881 y_hat_enet = predict(xgb_model, Xtest)
882 actual=y_test
883 pred=c(y_hat_enet)
884 rmse_m = rmse(y2,pred)
885 rmse_m
886 mape_m = mape(y2,pred)
887 mape_m
888
889
890
891 #####
892 nrow(trainset_new)
893 options(repr.plot.width=12,repr.plot.height=7)
894 hist_boxplot(trainset_new$Min.Delay,main="Delay time distribution",col="#9494b8",xlab="Delay time (in min")
895 boxplot.stats(trainset_new$Min.Delay)$stats
896 boxplot.stats(log(trainset_new$Min.Delay))$stats
897 x=which(trainset_new$Min.Delay %in% boxplot.stats(trainset_new$Min.Delay)$out)
898 length(x)
899
900 options(repr.plot.width=12,repr.plot.height=7)
901 hist_boxplot(log(trainset_new$Min.Delay),main="Log Delay time distribution",col="#9494b8",xlab="ln(delay")
902 boxplot.stats(log(trainset_new$Min.Delay))$stats
903 x=which(log(trainset_new$Min.Delay) %in% boxplot.stats(log(trainset_new$Min.Delay))$out)
904 length(x)
905 # Create a ggplot object
906 ggplot(trainset_new, aes(x = log(Min.Delay))) +
907   # Add histogram layer
908   geom_histogram(binwidth = 0.1, aes(y = ..density..), fill = "#9494b8", color = "black") +
909   # Add density curve layer
910   geom_density(color = "red", linetype = "solid", size = 1) +
911   # Add x-axis label
912   xlab("Log delay time") +
913   # Add y-axis label
914   ylab("Density") +
915   # Set theme
916   theme_minimal()
917
918 ggplot(data=trainset_new,aes(x=Min.Gap,y=log(Min.Delay)))+
919   geom_point()+
920   theme_minimal()
921
922 tabs(x="Min_Gap",y="log_Delay")
923
924 library(glmnet)
925 # Use cv.glmnet to perform cross-validation and select the optimal lambda value
926 #cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq)
927 set.seed(100)
928 cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)
929 # View the optimal lambda value
930 best_lambda2=cv_fits$lambda.1se
931 # Plot the cross-validation results
932 dev.off()
933 plot(cv_fit)
934
935 # Fit final model, get its sum of squared residuals and multiple R-squared
936 #model_cv = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda)
937 model_cv2 = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)
938 # Separate the response variable from the predictors
939 #y1 = df$Min.Delay
940 #y1 = df$Min.Delay
941 is.matrix(model_matrix)
942 dim(model_matrix)
943 # Create a sequence of lambda values to test
944 lambda_seq = 10^seq(10, -2, length = 100)
945
946
947
948 #For the test set
949 View(testset)
950 data2=testset[,-3]
951 data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
952 data2[,3]=as.numeric(data2[,3])
953 View(data2)
954
955 #model_matrix1 = model.matrix(log_delay~, data = data2)
956 model_matrix1 = model.matrix(log_delay~, data = data2)[,-1]
957 #model_matrix2 = model.matrix(log_delay~, data = data2)[,-1]
958 #model_matrix = model.matrix(Min.Delay~, data = df)
959
960 # Separate the response variable from the predictors
961 #y1 = data1$Min.Delay
962 #y1 = df$Min.Delay
963 is.matrix(model_matrix)
964 dim(model_matrix)
965 # Create a sequence of lambda values to test
966 lambda_seq = 10^seq(10, -2, length = 100)
967
968
969
970 #For the test set
971 View(testset)
972 data2=testset[,-3]
973 data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
974 data2[,3]=as.numeric(data2[,3])
975 View(data2)
976
977 #model_matrix1 = model.matrix(log_delay~, data = data2)
978 model_matrix1 = model.matrix(log_delay~, data = data2)[,-1]
979 #model_matrix2 = model.matrix(log_delay~, data = data2)[,-1]
980 #model_matrix = model.matrix(Min.Delay~, data = df)
981
982 # Separate the response variable from the predictors
983 #y1 = data1$Min.Delay
984 #y1 = df$Min.Delay
985 is.matrix(model_matrix)
986 dim(model_matrix)
987 # Create a sequence of lambda values to test
988 lambda_seq = 10^seq(10, -2, length = 100)
989
990
991
992 #For the test set
993 View(testset)
994 data2=testset[,-3]
995 data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
996 data2[,3]=as.numeric(data2[,3])
997 View(data2)
998
999 #model_matrix1 = model.matrix(log_delay~, data = data2)
1000 model_matrix1 = model.matrix(log_delay~, data = data2)[,-1]
1001 #model_matrix2 = model.matrix(log_delay~, data = data2)[,-1]
1002 #model_matrix = model.matrix(Min.Delay~, data = df)
1003
1004 # Separate the response variable from the predictors
1005 #y1 = data1$Min.Delay
1006 #y1 = df$Min.Delay
1007 is.matrix(model_matrix)
1008 dim(model_matrix)
1009 # Create a sequence of lambda values to test
1010 lambda_seq = 10^seq(10, -2, length = 100)
1011
1012
1013
1014 #For the test set
1015 View(testset)
1016 data2=testset[,-3]
1017 data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
1018 data2[,3]=as.numeric(data2[,3])
1019 View(data2)
1020
1021 #model_matrix1 = model.matrix(log_delay~, data = data2)
1022 model_matrix1 = model.matrix(log_delay~, data = data2)[,-1]
1023 #model_matrix2 = model.matrix(log_delay~, data = data2)[,-1]
1024 #model_matrix = model.matrix(Min.Delay~, data = df)
1025
1026 # Separate the response variable from the predictors
1027 #y1 = data1$Min.Delay
1028 #y1 = df$Min.Delay
1029 is.matrix(model_matrix)
1030 dim(model_matrix)
1031 # Create a sequence of lambda values to test
1032 lambda_seq = 10^seq(10, -2, length = 100)
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226

```

```

1227 - #####
1228 df_var_imp[order(-var_imp$importance),]
1229 View(df)
1230 nrow(df)
1231
1232 # Calculate number of variables to be removed
1233 vars_to_remove = df$Variables[36:56]
1234
1235 # Remove least important variables from data1
1236 dummy_data1= dummyVars(~ . , data = data1,sep = "")
1237 data1_encoded= predict(dummy_data1, newdata =data1)
1238 View(data1_encoded)
1239 data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
1240 View(data1_imp)
1241 colnames(data1_imp)
1242
1243 base_var=c("Route_NewRegular_and_limited service routes","DayMonday","MonthJanuary","Incident_NewCleaning")
1244 data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
1245 View(data1_imp)
1246 data1_impr
1247
1248 factor_cols = setdiff(names(data1_imp), num_cols)
1249 data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))
1250
1251 # Check the data types
1252 str(data1_impr)
1253 set.seed(100)
1254 indexes=sample(1:nrow(data1_impr), 0.1*nrow(data1_impr))
1255 data1_new=data1_impr[indexes,]
1256 # Define the tuning grid
1257 tune_grid = expand.grid(
1258   mtry=c(31, 25, 20, 15, 10),
1259   ntree=1000,
1260   nminobs=500
1261
1262 data2_impr=as.data.frame(data2_impr)
1263
1264 # Convert non-numeric columns to factors
1265 factor_cols = setdiff(names(data2_impr), num_cols)
1266 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1267
1268 y_hat_enet = predict(rf_model, data2_impr[,-7])
1269
1270 actual=y2
1271 pred=c(y_hat_enet )
1272 rmse_m = rmse(y2,pred)
1273 rmse_m
1274 mape_m = mape(y2, pred)
1275 mape_m
1276
1277 #####
1278 #Xgboost
1279 library(xgboost)
1280 Xtrain = xgb.DMatrix(data = as.matrix(data1[,-2]),label=y1)
1281 y_train = data1$log_delay
1282 Xtest = xgb.DMatrix(data = as.matrix(data2[,-2]),label=y2)
1283 y_test = data2$log_delay
1284 #Xgboost
1285 set.seed(100)
1286
1287 # get variable importance
1288 var_imp = varImp(xgb_model)
1289
1290 # print the variable importance
1291 print(var_imp)
1292
1293 # load ggplot2 library
1294 library(ggplot2)
1295
1296 # plot the variable importance
1297 ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
1298 geom_bar(stat = "identity", fill = "steelblue") +
1299 labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
1300 theme(plot.title = element_text(hjust = 0.5))
1301
1302 #####
1303 # Calculate number of variables to be removed
1304 vars_to_remove = df$Variables[23:56]
1305
1306 # Remove least important variables from data1
1307 dummy_data1= dummyVars(~ . , data = data1,sep = "")
1308 data1_encoded= predict(dummy_data1, newdata =data1)
1309 View(data1_encoded)
1310 data1_impr
1311 base_var=c("Route_NewRegular_and_limited service routes","DayMonday","MonthJanuary","Incident_NewCleaning")
1312 data1_impr = data1_impr[, !(colnames(data1_impr) %in% base_var)]
1313 View(data1_impr)
1314 str(data1_impr)
1315 data1_impr
1316
1317 # Convert non-numeric columns to factors
1318 factor_cols = setdiff(names(data2_impr), num_cols)
1319 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1320
1321 y_hat_enet = predict(rf_model, data2_impr[,-7])
1322
1323 actual=y2
1324 pred=c(y_hat_enet )
1325 rmse_m = rmse(y2,pred)
1326 rmse_m
1327 mape_m = mape(y2, pred)
1328 mape_m
1329
1330 #####
1331 #Xgboost
1332 library(xgboost)
1333 Xtrain = xgb.DMatrix(data = as.matrix(data1[,-2]),label=y1)
1334 y_train = data1$log_delay
1335 Xtest = xgb.DMatrix(data = as.matrix(data2[,-2]),label=y2)
1336 y_test = data2$log_delay
1337 #Xgboost
1338 set.seed(100)
1339
1340 # get variable importance
1341 var_imp = varImp(xgb_model)
1342
1343 # print the variable importance
1344 print(var_imp)
1345
1346 # load ggplot2 library
1347 library(ggplot2)
1348
1349 # plot the variable importance
1350 ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
1351 geom_bar(stat = "identity", fill = "steelblue") +
1352 labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
1353 theme(plot.title = element_text(hjust = 0.5))
1354
1355 #####
1356 # Calculate number of variables to be removed
1357 vars_to_remove = df$Variables[23:56]
1358
1359 # Remove least important variables from data1
1360 dummy_data1= dummyVars(~ . , data = data1,sep = "")
1361 data1_encoded= predict(dummy_data1, newdata =data1)
1362 View(data1_encoded)
1363 data1_impr
1364 colnames(data1_impr)
1365 base_var=c("Route_NewRegular_and_limited service routes","DayMonday","MonthJanuary","Incident_NewCleaning")
1366 data1_impr = data1_impr[, !(colnames(data1_impr) %in% base_var)]
1367 View(data1_impr)
1368 str(data1_impr)
1369 data1_impr
1370
1371 # Convert non-numeric columns to factors
1372 factor_cols = setdiff(names(data2_impr), num_cols)
1373 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1374
1375 Xtrain = xgb.DMatrix(data = as.matrix(data1_impr[,-4]),label=y1)
1376 y_train = data1$log_delay
1377
1378 set.seed(100)
1379 xgb_model = train(Xtrain,
1380   y_train,
1381   method = "xgbTree",
1382   objective = "reg:squarederror",
1383   trControl = trainControl(method = "cv",
1384     number = 5,
1385     verboseIter = TRUE),
1386
1387 tuneGrid = expand.grid(nrounds = c(500,1000),
1388   eta=0.1,
1389   max_depth = c(2,4,6),
1390   colsample_bytree = c(0.5,0.6),
1391   subsample = c(0.5,0.6),
1392   gamma=0.1,
1393   min_child_weight = 1
1394
1395 plot(xgb_model)
1396 y_hat_enet = predict(xgb_model, Xtrain)
1397 actual=y1
1398 pred=c(y_hat_enet)
1399 d_actual=trainset$newMin.Delay
1400 d_pred=as.integer(exp(pred))
1401 rmse_m = rmse(d_actual, d_pred)
1402 rmse_m
1403
1404 #For the test set
1405 y_hat_enet = predict(xgb_model, Xtest)
1406 actual=y2
1407 pred=c(y_hat_enet)
1408 d_actual=testset$Min.Delay
1409 d_pred=as.integer(exp(pred))
1410 rmse_m = rmse(d_actual, d_pred)
1411 rmse_m
1412
1413 #####
1414 # get variable importance
1415 var_imp = varImp(xgb_model)
1416
1417 # print the variable importance
1418 print(var_imp)
1419
1420 # load ggplot2 library
1421 library(ggplot2)
1422
1423 # plot the variable importance
1424 ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
1425 geom_bar(stat = "identity", fill = "steelblue") +
1426 labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
1427 theme(plot.title = element_text(hjust = 0.5))
1428
1429 #####
1430 # Calculate number of variables to be removed
1431 vars_to_remove = df$Variables[23:56]
1432
1433 # Remove least important variables from data1
1434 dummy_data1= dummyVars(~ . , data = data1,sep = "")
1435 data1_encoded= predict(dummy_data1, newdata =data1)
1436 View(data1_encoded)
1437 data1_impr
1438 colnames(data1_impr)
1439 base_var=c("Route_NewRegular_and_limited service routes","DayMonday","MonthJanuary","Incident_NewCleaning")
1440 data1_impr = data1_impr[, !(colnames(data1_impr) %in% base_var)]
1441 View(data1_impr)
1442 str(data1_impr)
1443 data1_impr
1444
1445 # Convert non-numeric columns to factors
1446 factor_cols = setdiff(names(data2_impr), num_cols)
1447 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1448
1449 Xtrain = xgb.DMatrix(data = as.matrix(data1_impr[,-4]),label=y1)
1450 y_train = data1$log_delay
1451
1452 set.seed(100)
1453 xgb_model = train(Xtrain,
1454   y_train,
1455   method = "xgbTree",
1456   objective = "reg:squarederror",
1457   trControl = trainControl(method = "cv",
1458     number = 5,
1459     verboseIter = TRUE),
1460
1461 tuneGrid = expand.grid(nrounds = c(500,1000),
1462   eta=0.1,
1463   max_depth = c(2,4,6),
1464   colsample_bytree = c(0.5,0.6),
1465   subsample = c(0.5,0.6),
1466   gamma=0.1,
1467   min_child_weight = 1
1468
1469 plot(xgb_model)
1470 y_hat_enet = predict(xgb_model, Xtrain)
1471 actual=y1
1472 pred=c(y_hat_enet)
1473 rmse_m = rmse(y1,pred)
1474 rmse_m
1475
1476 #For the test set
1477 # Remove least important variables from data2
1478
1479 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1480 data2_encoded= predict(dummy_data2, newdata =data2)
1481 View(data2_encoded)
1482 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1483 View(data2_impr)
1484 str(data2_impr)
1485 data2_impr
1486
1487 # Convert non-numeric columns to factors
1488 factor_cols = setdiff(names(data2_impr), num_cols)
1489 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1490
1491 Xtest = xgb.DMatrix(data = data2_impr[,-4],label=y2)
1492 y_test = data2$log_delay
1493
1494 y_hat_enet = predict(xgb_model, Xtest)
1495 actual=y2
1496 pred=c(y_hat_enet)
1497 rmse_m = rmse(y1,pred)
1498 rmse_m
1499
1500 #For the test set
1501 # Remove least important variables from data2
1502
1503 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1504 data2_encoded= predict(dummy_data2, newdata =data2)
1505 View(data2_encoded)
1506 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1507 View(data2_impr)
1508 str(data2_impr)
1509 data2_impr
1510
1511 # Convert non-numeric columns to factors
1512 factor_cols = setdiff(names(data2_impr), num_cols)
1513 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1514
1515 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1516 y_train = data2$log_delay
1517
1518 set.seed(100)
1519 xgb_model = train(Xtrain,
1520   y_train,
1521   method = "xgbTree",
1522   objective = "reg:squarederror",
1523   trControl = trainControl(method = "cv",
1524     number = 5,
1525     verboseIter = TRUE),
1526
1527 tuneGrid = expand.grid(nrounds = c(500,1000),
1528   eta=0.1,
1529   max_depth = c(2,4,6),
1530   colsample_bytree = c(0.5,0.6),
1531   subsample = c(0.5,0.6),
1532   gamma=0.1,
1533   min_child_weight = 1
1534
1535 plot(xgb_model)
1536 y_hat_enet = predict(xgb_model, Xtrain)
1537 actual=y1
1538 pred=c(y_hat_enet)
1539 rmse_m = rmse(y1,pred)
1540 rmse_m
1541
1542 #For the test set
1543 # Remove least important variables from data2
1544
1545 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1546 data2_encoded= predict(dummy_data2, newdata =data2)
1547 View(data2_encoded)
1548 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1549 View(data2_impr)
1550 str(data2_impr)
1551 data2_impr
1552
1553 # Convert non-numeric columns to factors
1554 factor_cols = setdiff(names(data2_impr), num_cols)
1555 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1556
1557 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1558 y_train = data2$log_delay
1559
1560 set.seed(100)
1561 xgb_model = train(Xtrain,
1562   y_train,
1563   method = "xgbTree",
1564   objective = "reg:squarederror",
1565   trControl = trainControl(method = "cv",
1566     number = 5,
1567     verboseIter = TRUE),
1568
1569 tuneGrid = expand.grid(nrounds = c(500,1000),
1570   eta=0.1,
1571   max_depth = c(2,4,6),
1572   colsample_bytree = c(0.5,0.6),
1573   subsample = c(0.5,0.6),
1574   gamma=0.1,
1575   min_child_weight = 1
1576
1577 plot(xgb_model)
1578 y_hat_enet = predict(xgb_model, Xtrain)
1579 actual=y1
1580 pred=c(y_hat_enet)
1581 rmse_m = rmse(y1,pred)
1582 rmse_m
1583
1584 #For the test set
1585 # Remove least important variables from data2
1586
1587 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1588 data2_encoded= predict(dummy_data2, newdata =data2)
1589 View(data2_encoded)
1590 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1591 View(data2_impr)
1592 str(data2_impr)
1593 data2_impr
1594
1595 # Convert non-numeric columns to factors
1596 factor_cols = setdiff(names(data2_impr), num_cols)
1597 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1598
1599 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1600 y_train = data2$log_delay
1601
1602 set.seed(100)
1603 xgb_model = train(Xtrain,
1604   y_train,
1605   method = "xgbTree",
1606   objective = "reg:squarederror",
1607   trControl = trainControl(method = "cv",
1608     number = 5,
1609     verboseIter = TRUE),
1610
1611 tuneGrid = expand.grid(nrounds = c(500,1000),
1612   eta=0.1,
1613   max_depth = c(2,4,6),
1614   colsample_bytree = c(0.5,0.6),
1615   subsample = c(0.5,0.6),
1616   gamma=0.1,
1617   min_child_weight = 1
1618
1619 plot(xgb_model)
1620 y_hat_enet = predict(xgb_model, Xtrain)
1621 actual=y1
1622 pred=c(y_hat_enet)
1623 rmse_m = rmse(y1,pred)
1624 rmse_m
1625
1626 #For the test set
1627 # Remove least important variables from data2
1628
1629 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1630 data2_encoded= predict(dummy_data2, newdata =data2)
1631 View(data2_encoded)
1632 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1633 View(data2_impr)
1634 str(data2_impr)
1635 data2_impr
1636
1637 # Convert non-numeric columns to factors
1638 factor_cols = setdiff(names(data2_impr), num_cols)
1639 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1640
1641 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1642 y_train = data2$log_delay
1643
1644 set.seed(100)
1645 xgb_model = train(Xtrain,
1646   y_train,
1647   method = "xgbTree",
1648   objective = "reg:squarederror",
1649   trControl = trainControl(method = "cv",
1650     number = 5,
1651     verboseIter = TRUE),
1652
1653 tuneGrid = expand.grid(nrounds = c(500,1000),
1654   eta=0.1,
1655   max_depth = c(2,4,6),
1656   colsample_bytree = c(0.5,0.6),
1657   subsample = c(0.5,0.6),
1658   gamma=0.1,
1659   min_child_weight = 1
1660
1661 plot(xgb_model)
1662 y_hat_enet = predict(xgb_model, Xtrain)
1663 actual=y1
1664 pred=c(y_hat_enet)
1665 rmse_m = rmse(y1,pred)
1666 rmse_m
1667
1668 #For the test set
1669 # Remove least important variables from data2
1670
1671 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1672 data2_encoded= predict(dummy_data2, newdata =data2)
1673 View(data2_encoded)
1674 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1675 View(data2_impr)
1676 str(data2_impr)
1677 data2_impr
1678
1679 # Convert non-numeric columns to factors
1680 factor_cols = setdiff(names(data2_impr), num_cols)
1681 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1682
1683 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1684 y_train = data2$log_delay
1685
1686 set.seed(100)
1687 xgb_model = train(Xtrain,
1688   y_train,
1689   method = "xgbTree",
1690   objective = "reg:squarederror",
1691   trControl = trainControl(method = "cv",
1692     number = 5,
1693     verboseIter = TRUE),
1694
1695 tuneGrid = expand.grid(nrounds = c(500,1000),
1696   eta=0.1,
1697   max_depth = c(2,4,6),
1698   colsample_bytree = c(0.5,0.6),
1699   subsample = c(0.5,0.6),
1700   gamma=0.1,
1701   min_child_weight = 1
1702
1703 plot(xgb_model)
1704 y_hat_enet = predict(xgb_model, Xtrain)
1705 actual=y1
1706 pred=c(y_hat_enet)
1707 rmse_m = rmse(y1,pred)
1708 rmse_m
1709
1710 #For the test set
1711 # Remove least important variables from data2
1712
1713 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1714 data2_encoded= predict(dummy_data2, newdata =data2)
1715 View(data2_encoded)
1716 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1717 View(data2_impr)
1718 str(data2_impr)
1719 data2_impr
1720
1721 # Convert non-numeric columns to factors
1722 factor_cols = setdiff(names(data2_impr), num_cols)
1723 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1724
1725 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1726 y_train = data2$log_delay
1727
1728 set.seed(100)
1729 xgb_model = train(Xtrain,
1730   y_train,
1731   method = "xgbTree",
1732   objective = "reg:squarederror",
1733   trControl = trainControl(method = "cv",
1734     number = 5,
1735     verboseIter = TRUE),
1736
1737 tuneGrid = expand.grid(nrounds = c(500,1000),
1738   eta=0.1,
1739   max_depth = c(2,4,6),
1740   colsample_bytree = c(0.5,0.6),
1741   subsample = c(0.5,0.6),
1742   gamma=0.1,
1743   min_child_weight = 1
1744
1745 plot(xgb_model)
1746 y_hat_enet = predict(xgb_model, Xtrain)
1747 actual=y1
1748 pred=c(y_hat_enet)
1749 rmse_m = rmse(y1,pred)
1750 rmse_m
1751
1752 #For the test set
1753 # Remove least important variables from data2
1754
1755 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1756 data2_encoded= predict(dummy_data2, newdata =data2)
1757 View(data2_encoded)
1758 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1759 View(data2_impr)
1760 str(data2_impr)
1761 data2_impr
1762
1763 # Convert non-numeric columns to factors
1764 factor_cols = setdiff(names(data2_impr), num_cols)
1765 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1766
1767 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1768 y_train = data2$log_delay
1769
1770 set.seed(100)
1771 xgb_model = train(Xtrain,
1772   y_train,
1773   method = "xgbTree",
1774   objective = "reg:squarederror",
1775   trControl = trainControl(method = "cv",
1776     number = 5,
1777     verboseIter = TRUE),
1778
1779 tuneGrid = expand.grid(nrounds = c(500,1000),
1780   eta=0.1,
1781   max_depth = c(2,4,6),
1782   colsample_bytree = c(0.5,0.6),
1783   subsample = c(0.5,0.6),
1784   gamma=0.1,
1785   min_child_weight = 1
1786
1787 plot(xgb_model)
1788 y_hat_enet = predict(xgb_model, Xtrain)
1789 actual=y1
1790 pred=c(y_hat_enet)
1791 rmse_m = rmse(y1,pred)
1792 rmse_m
1793
1794 #For the test set
1795 # Remove least important variables from data2
1796
1797 dummy_data2= dummyVars(~ . , data = data2,sep = "")
1798 data2_encoded= predict(dummy_data2, newdata =data2)
1799 View(data2_encoded)
1800 data2_impr = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
1801 View(data2_impr)
1802 str(data2_impr)
1803 data2_impr
1804
1805 # Convert non-numeric columns to factors
1806 factor_cols = setdiff(names(data2_impr), num_cols)
1807 data2_impr[factor_cols] = apply(data2_impr[factor_cols], 2, function(x) as.factor(x))
1808
1809 Xtrain = xgb.DMatrix(data = as.matrix(data2_impr[,-4]),label=y1)
1810 y_train = data2$log_delay
1811
1812 set.seed(100)
1813 xgb_model = train(Xtrain,
1814   y_train,
1815   method = "xgbTree",
1816   objective = "reg:squarederror",
1817   trControl = trainControl(method = "cv",
1818     number = 5,
1819     verboseIter = TRUE),
1820
1821 tuneGrid = expand.grid(nrounds = c(500,1000),
1822   eta=0.1,
1823   max_depth = c(2,4,6),
1824   colsample_bytree = c(0.5,0.6),
1825   subsample = c(0.5,0.6),
1826   gamma=0.1,
1827   min_child_weight = 1
1828
1829 plot(xgb_model)
1830 y_hat_enet = predict(xgb_model, Xtrain)
1831 actual=y1
1832 pred=c(y_hat_enet)
1833 rmse_m = rmse(y1,pred)
1834 rmse_m

```