STATS 101C Final Project

Limit DNE: Hana Yerin Lim, Yanhua Lin, Yingzhen Zhao

December 2020

1 Introduction

In this project, we aim to find the optimal factors to predict the percentage change in views of a video between the second and sixth hour since its publishing. The training data set includes 7242 videos, 258 predictors and 1 response variable called *growth_2_6*. There are 3105 videos in the test data set needed to predicted.

Our goal is to closely predict the true value of *growth_2_6* in the test data set. In order evaluate this problem, we need to minimize the root mean squared error (RMSE) metric measuring the distance between our predicted growth percentage and the true growth percentage, which is computed as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(g_i - \hat{g}_i)^2}$$

2 Methodology

This section discusses data preprocessing and the results we obtained from establishing different models.

2.1 Preprocessing of the data

- Creating additional columns

  From the feature description, there were several binary variables with different levels and one categorical variable called *PublishedDate*. We created several additional features regrading to these columns. Variable names and description are as follow:

  1. *Num_Subscribers_Base_high*, *Num_Views_Base_high*, *avg_growth_high*, *count_vids_high*:
     In addition to the existing levels of number of subscribers, number of total views, average growth from 2 to 6 hour, and the number of other videos that consisted of low, low mid, and mid high, we added additional high levels of corresponding data.
  2. *Days*: We split the *PublishedDate* column, which contains the date and time of the video published on YouTube from April to September, into date and time columns respectively. We pulled the date and used 4/1/2020 to be the first day in order to observe how the growth rate changes day by day.
  3. *Hours*: From the *PublishedDate* column, we pulled the time and convert it into hours in decimal form from midnight.
  4. *midnight*, *morning*, *noon*, *afternoon*, *evening*: Divided the 24 hour a day into 5 chunks and made them into binary columns.

  (For more details of the creation, please see the appendix on pg.1-2)

- Removing not important and highly correlated predictors

An important factor to consider is the existence of multicollinearity. This may impair the accuracy of the prediction and therefore needs to be avoided. First, we remove columns with zero variances, since all values within each of these columns are identical and they will not contribute to the prediction. Then, in order to remove highly correlated predictors, we created a while loop and we tested with different values of correlation thresholds on numeric variables and found out that 0.8 was the best threshold that kept enough amount of predictors. Although *views_2_hours* and *Duration* columns were removed after cleaning up the predictors, they are added back because they are considered to be important predictors from the description. (For more details, please see appendix on pg. 2)

After creating and cleaning the data set, we now have 7242 observations and 183 variables with 182 predictors and 1 response.

2.2 Description of the models

- Model and Predictors Selection

In order to compare the RMSE scores (calculated) from different models, we partitioned the data set into training and validation sets by 0.8/0.2 ratio. We fit LASSO regression, boosting, bagging and random forests to the training set using 182 predictors. Methods and parameters used for these models are described as follow:

LASSO: Our team started off by trying LASSO prediction because of its improvement on prediction accuracy and interpretability of the statistical model. We selected the best value for lambda using K-fold cross-validation using 10 folds. Despite obtaining the best predictor selections with the best value of λ of 0.010, the RMSE score only turned out to be 1.650821. (reference appendix on pg.4)

Boosting: We preform boosting on the training set with 1000 trees for a range of values of the shrinkage parameter λ. We then obtained a best λ 0.066 and a smallest RMSE 1.628784, which is slightly better than while using LASSO regression. (reference appendix on pg.4-5)

Bagging: Afterwards, we performed bagging method using the out-of-bag error estimate with mtry value of 182(numbers of predictors) and ntree value of 1000. We got the RMSE of 1.475989, which is a big improvement compared to LASSO and boosting. We then hold onto this method to compare with the random forest method, which utilizes even a smaller number of variables.

Random Forest: Performing a random forest is similar to bagging, as we fixed all the values of parameters and using out-of-bag error estimate except we used 60 (numbers of predictors/3 = 182/3) for mtry argument. The testing set RMSE associated with the random forest for regression tree is 1.478593 which is similar to what we obtained from bagging but much smaller than that of LASSO and boosting.

⋆ Table 1: RMSEs obtained from 4 models (All based on predictors p = 182)

Model	RMSE	Comment
LASSO	1.650821	Best $\lambda = 0.010$
Boosting	1.628784	Best $\lambda = 0.066$
Bagging	1.475989	mtry = 182
RandomForests	1.478593	mtry = 182 / 3 = 60

(For more details of the coding, please see appendix on pg. 4-5)

Different RMSEs associated with different models are shown in Table 1. Since the RMSEs obtained from bagging and random forest based on 182 predictors were very similar, we decided to do more comparison with these two models. We extracted all the predictors in the order of importance using the importance() function and the dplyr arrange function. We tried different range of thresholds from 3 to 11, and we noticed that 8, 9, and 10 were the best among this range. We then performed both bagging and random forest models once again to experiment with the best threshold values (8,9,10) of variable importance to get the most significant predictors. The results of RMSEs are summarized in Table 2.

⋆ Table 2: Bagging vs RandomForest with different number of predictors

Model	Threshold %IncMSE >	Number of Predictors p	mtry	RMSE
Bagging	10	28	28	1.434410
Bagging	9	31	31	1.433804
Bagging	8	32	32	1.438456
RandomForest	10	33	11	1.423562
RandomForest	9	35	11	1.423923
RandomForest	8	41	13	1.421445

(For more details of the coding, please see appendix on pg. 5-8)

After analyzing the results, we concluded that random forest performed better improvement over bagging in this case. Thus, we decided to choose random forest as our final model and worked further on improving our final RMSE score using optimal arguments and different subsets of predictors.

- Processing Random Forests

Having decided to use Random Forest as our final model, we tried to find the optimal subset of predictors. After running the initial randomForest() with mtry of p/3 and listing 182 predictors in the order of the importance, we not only tested different values of threshold, but also further tuned the parameters to find the optimized mtry and sufficiently large enough ntree to stabilize from Out-of-bag error, which would prevent our model from overfitting. The results of RMSEs with corresponding threshold and mtry values are shown in Table 3.

⋆ Table 3: The results of processing Random Forest Model

Model	Threshold %IncMSE >	Number of Predictors p	mtry	RMSE
RandomForest	8	41	p/3 = 13	1.421445
RandomForest	8	41	14	1.422817
RandomForest	8	41	12	1.425424
RandomForest	9	35	p/3 = 11	1.423923
RandomForest	9	35	12	1.422035
RandomForest	9	35	10	1.421007
RandomForest	10	33	p/3 = 11	1.423562
RandomForest	10	33	12	**1.412062**
RandomForest	10	33	10	**1.414207**
RandomForest	11	29	p/3 = 10	1.424145

(For more details of the coding, please see appendix on pg. 6-8)

We can observe from Table 3 that the model with 33 predictors and mtry of 12 generates the best RMSE.

3 Results

Using our final model as the random forest with 33 predictors and mtry of 12 with ntree of 1000 yielded our best evaluation metric value of 1.35764 on Kaggle public leaderboard. The model that is obtained under many experiments performs well since it generated the lowest RMSE. When we applied our final model to the test data, we not only beat all of the benchmarks, but also outperformed other teams as we placed first in the competition.

4 Conclusions

We believe that there are several reasons to the successful outcome. Not only we had the ability to choose the right choice of the model, select important and necessary features, and remove highly correlated predictors, but also we were able to be creative and add additional features into our dataset.

We approached our analysis by crossing out the worse models. Despite the interpretability, LASSO wasn't the final selection because the assumption is based on linear regression and the response are not strongly correlated with other variables. The weakness of boosting method is its sensitivity to overfitting of the data containing too many noises and difficulty of tuning compared to random forest. Lastly, we had to take the number of important features and the value of mtry into account in order to evaluate the better method between bagging and random forest, which turned out to be random forest model (Table 2).

After trying different types of models, we were confident that random forest worked the best after narrowing down our methods (Table 1 and Table 2). After selecting random forest model to dive into further exploration, our remaining task was to tune random forest with different values of predictors and mtry to find the best condition for the lowest RMSE value (Table 3).

On top of finding the right threshold value to reduce into a major set of predictors and removing highly correlated variables, another way to strengthen our model was the additional variables of the binary columns for high levels of the existing binary columns and time intervals, hours and Days columns. In fact, 8 of the features out of 11 new features were part of the final 33 predictors.

Compared to the Kaggle public leaderboard score of 1.35764, we yielded 1.37819 from the private leaderboard. Adding the remaining test data caused a slight RMSE change, but we consider this two scores are similar. This is because random forest method with enough important predictors and sufficiently large value of trees limits overfitting as well as errors due to bias and therefore yield useful results.

On the other hand, there are still some possible ways to improve our model. For example, there's also a lot of inspiration for creating new variables based on the existing columns. We can also consider performing some of the advanced techniques such as oversampling and undersampling, and testing wider range of ML techniques that we haven't seen in class, such as XGBoosting method.

Overall, we are satisfied with our results, but we acknowledge that there could a possibility that we could further improve our model to get even the lower prediction error rate.

5 Statement of Contributions

All the members collaborated on every parts of the project.
Predictor Selection: Yanhua Lin, Hana Yerin Lim, Yingzhen Zhao
Processing Methods: Hana Yerin Lim, Yanhua Lin, Yingzhen Zhao
Report: Yingzhen Zhao, Hana Yerin Lim, Yanhua Lin

Appendix

Loading data and create additional columns

```
library(tidyverse)
library(readr)
library(readxl)
library(randomForest)
library(lubridate)
library(caret)

# load data
traincsv <- read.csv("training.csv")
test <- read.csv("test.csv")

# processing data of PublishedDate column
training <- traincsv[ , -260]
date_col <- training$PublishedDate
s <- unlist(strsplit(as.character(date_col), split = " "))
time <- matrix(s, ncol = 2, byrow = TRUE)[ , 2]
t <- unlist(strsplit(time, split = ":"))
h_n_m <- matrix(as.numeric(t), ncol = 2, byrow = TRUE)
hours <- h_n_m[ ,1] + h_n_m[ ,2] / 60
Published_date <- matrix(s, ncol = 2, byrow = TRUE)[ ,1]
Days <- as.numeric(mdy(Published_date) - mdy("3/31/2020"))

# create additional columns regarding to binary variables
training$Num_Subscribers_Base_high <- as.numeric((training$Num_Subscribers_Base_low == 0) &
                                             (training$Num_Subscribers_Base_low_mid == 0) &
                                            (training$Num_Subscribers_Base_mid_high == 0))

training$Num_Views_Base_high <- as.numeric((training$Num_Views_Base_low == 0) &
                                       (training$Num_Views_Base_low_mid == 0) &
                                      (training$Num_Views_Base_mid_high == 0))

training$avg_growth_high <- as.numeric((training$avg_growth_low == 0) &
                                   (training$avg_growth_low_mid == 0) &
                                  (training$avg_growth_mid_high == 0))

training$count_vids_high <- as.numeric((training$count_vids_low == 0) &
                                   (training$count_vids_low_mid == 0) &
                                  (training$count_vids_mid_high == 0))
```

```r
# create additional columns regarding to PublishedDate column
training$midnight <- as.numeric((hours >= 0 & hours < 5) |
                                  (hours >= 23 & hours < 24))
training$morning <- as.numeric(hours >= 5 & hours < 12)
training$noon <- as.numeric(hours >= 12 & hours < 13)
training$afternoon <- as.numeric(hours >= 13 & hours < 20)
training$night <- as.numeric(hours >= 20 & hours < 23)
training$hours <- hours
training$days <- Days

Binary_variable <- training[ ,248:268]
Binary_variable_names <- colnames(Binary_variable)

# factorize binary columns
for (i in 248:268) {
  training[ , i] <- factor(training[ , i])
}

# complete training data
training$growth_2_6 <- traincsv$growth_2_6

dim(training)
```

- After adding necessary columns, we now have 7242 observations and 271 columns.

Removing not important or highly correlated columns

```r
filtered <- training[ , -c(1, 2, 248:268, 271)] # remove columns of 'id',
                                                # 'date', 'binary', 'response'
filtered <- filtered[ , 247:1]
variances <- apply(filtered, 2, var)
filtered_variances <- unname(which(variances == 0))
filtered <- filtered[ , - filtered_variances]  # remove columns with variance = 0

# remove predictors highly correlated
save_features <- character()
while(ncol(filtered) > 2) {
  correlation <- cor(filtered[ , 1], filtered[ , 2 : ncol(filtered)])
  high_correlated <- names(which(correlation[1, ] > 0.8))
  left_features <- colnames(filtered)[!(colnames(filtered) %in% high_correlated)]
  left_features <- left_features[-1]
  save_features <- c(save_features, names(filtered[1]))
  filtered <- filtered[ ,left_features]
}
train_left <- as.data.frame(training[ ,save_features])

# adding important columns back
train_left$views_2_hours <- training$views_2_hours
train_left$Duration <- training$Duration

Binary_columns <- training[ , 248:268]
train_left <- cbind(train_left, Binary_columns)
```

```r
train_left$growth_2_6 <- training$growth_2_6

dim(train_left)
```

- After removing not important or highly correlated columns, we now have 7242 observations and 183 variables with 182 predictors and 1 response.

Processing and cleaning the test data set

```r
test$Num_Subscribers_Base_high <- as.numeric((test$Num_Subscribers_Base_low == 0) &
                                          (test$Num_Subscribers_Base_low_mid == 0) &
                                         (test$Num_Subscribers_Base_mid_high == 0))

test$Num_Views_Base_high <- as.numeric((test$Num_Views_Base_low == 0) &
                                    (test$Num_Views_Base_low_mid == 0) &
                                   (test$Num_Views_Base_mid_high == 0))

test$avg_growth_high <- as.numeric((test$avg_growth_low == 0) &
                                (test$avg_growth_low_mid == 0) &
                               (test$avg_growth_mid_high == 0))

test$count_vids_high <- as.numeric((test$count_vids_low == 0) &
                                (test$count_vids_low_mid == 0) &
                               (test$count_vids_mid_high == 0))

date_col1 <- test$PublishedDate
s1 <- unlist(strsplit(as.character(date_col1), split = " "))
time1 <- matrix(s1, ncol = 2, byrow = TRUE)[ , 2]
t1 <- unlist(strsplit(time1, split = ":"))
h_n_m1 <- matrix(as.numeric(t1), ncol = 2, byrow = TRUE)
hours1 <- h_n_m1[ ,1] + h_n_m1[ ,2] / 60
Published_date <- matrix(s1, ncol = 2, byrow = TRUE)[ ,1]
Days <- as.numeric(mdy(Published_date) - mdy("3/31/2020"))

test$midnight <- as.numeric((hours1 >= 0 & hours1 < 5) |
                              (hours1 >= 23 & hours1 < 24))
test$morning <- as.numeric(hours1 >= 5 & hours1 < 12)
test$noon <- as.numeric(hours1 >= 12 & hours1 < 13)
test$afternoon <- as.numeric(hours1 >= 13 & hours1 < 20)
test$night <- as.numeric(hours1 >= 20 & hours1 < 23)
test$hours <- hours1
test$days <- Days

# factorize binary columns
for (i in 248:268) {
  test[ , i] <- factor(test[ , i])
}

dim(test)
```

- After processing and cleaning the test data set, we now have 3105 observations and 270 columns.

Creat a training set containing a random sample of 80% observations of the origionl data, and a validation set containing the remaining observations.

```r
set.seed(1000)
i <- createDataPartition(train_left$growth_2_6, p = 0.8, list = FALSE)
train <- train_left[i, ]
test1 <- train_left[-i, ]
dim(train)
dim(test1)
dim(test)
```

Fit a LASSO regression

```r
set.seed(1000)
library(glmnet)
x <- model.matrix(growth_2_6 ~ ., data = train)[ , -1]
y <- train$growth_2_6

x_test <- model.matrix(growth_2_6 ~ ., data = test1)[ , -1]

grid <- 10 ^ seq(1, -3, length = 100)

lasso_model <- glmnet(x, y, family = "gaussian", alpha = 1,
                      lambda = grid, standardize = TRUE)

# Select the best value for lambda using K-fold cross-validation.
cv_lasso <- cv.glmnet(x, y, family = "gaussian", alpha = 1,
                      lambda = grid, standardize = TRUE, nfolds = 10)

# Retrieve the actual best value of lambda.
best_lambda_cv_l <- cv_lasso$lambda.min
best_lambda_cv_l

lasso_pred <- predict(lasso_model, s = best_lambda_cv_l, x_test)
lasso_rmse <- sqrt(mean((test1$growth_2_6 - lasso_pred) ^ 2))
lasso_rmse
```

- Best lambda value is 0.01023531, with root mean square error 1.650821.

Fit a boosting model

```r
library(gbm)
test_MSE <- numeric(100)
i.exp <- seq(-10, -1, length = 100)
lambda <- 10 ^ i.exp

for (i in 1:100) {
  boosting <- gbm(growth_2_6 ~ . , data = train,
                  distribution = "gaussian",
                  n.trees = 1000, shrinkage = lambda[i])

  yhat_boost <- predict(boosting, newdata = test1, n.trees = 1000)
  test_MSE[i] <- mean((test1$growth_2_6 - yhat_boost) ^ 2)
}

boosting_best <- gbm(growth_2_6 ~ . , data = train,
                     distribution = "gaussian",
                     n.trees = 1000,
                     shrinkage = lambda[which.min(test_MSE)],
                     cv.folds = 10)

yhat_boost <- predict(boosting_best, newdata = test1, n.trees = 1000)
boost_rmse <- sqrt(mean((test1$growth_2_6 - yhat_boost) ^ 2))
boost_rmse
```

- With 10 fold cross-validation and best lambda value 0.06579332, our root mean square error is 1.600489.

Fit a bagging model

```
bagging <- randomForest(growth_2_6 ~ . ,
                        data = train, mtry = ncol(train) - 1,
                        ntree = 1000, importance = TRUE)

bag_pred_1 <- predict(bagging, test1)
bag_rmse_1 <- sqrt(mean((test1$growth_2_6 - bag_pred_1) ^ 2))
bag_rmse_1
```

- RMSE obtained is 1.475989.

```
bag_im <-importance(bagging)

new <- rownames(bag_im)[which(abs(bag_im[,1]) > 10)]
newdata <- train[, c(new, "growth_2_6")]
dim(newdata)

bag <- randomForest(growth_2_6 ~ . , data = newdata,
                    mtry = ncol(newdata) - 1, ntree = 1000, importance = TRUE)
bag_pred <- predict(bag, test1)
bag_rmse <- sqrt(mean((test1$growth_2_6 - bag_pred) ^ 2))
bag_rmse
```

- With 28 predictors, RMSE obtained is 1.43441.

```
new_bag_9 <- rownames(bag_im)[which(abs(bag_im[,1]) > 9)]
newdata_bag_9 <- train[, c(new_bag_9, "growth_2_6")]
dim(newdata_bag_9)

bag_bag_9 <- randomForest(growth_2_6 ~ . , data = newdata_bag_9,
                    mtry = ncol(newdata_bag_9) - 1, ntree = 1000, importance = TRUE)
bag_pred_bag_9 <- predict(bag_bag_9, test1)
bag_rmse_bag_9 <- sqrt(mean((test1$growth_2_6 - bag_pred_bag_9) ^ 2))
bag_rmse_bag_9
```

- With 31 predictors, RMSE obtained is 1.433804.

```
new_bag_8 <- rownames(bag_im)[which(abs(bag_im[,1]) > 8)]
newdata_bag_8 <- train[, c(new_bag_8, "growth_2_6")]
dim(newdata_bag_8)
```

```r
bag_bag_8 <- randomForest(growth_2_6 ~ . , data = newdata_bag_8,
                    mtry = ncol(newdata_bag_8) - 1, ntree = 1000, importance = TRUE)
bag_pred_bag_8 <- predict(bag_bag_8, test1)
bag_rmse_bag_8 <- sqrt(mean((test1$growth_2_6 - bag_pred_bag_8) ^ 2))
bag_rmse_bag_8
```

- With 32 predictos, RMSE obtained is 1.438456.

Fit a random forest model

```r
rf_fit <- randomForest(growth_2_6 ~ . , data = train,
                       mtry = floor((ncol(train) - 1) / 3),
                       ntree = 1000, importance = TRUE)

rf_pred <- predict(rf_fit, test1)
RMSE_rf <- sqrt(mean((test1$growth_2_6 - rf_pred) ^ 2))
RMSE_rf
```

- RMSE obtained is 1.478593.

```r
varImp(rf_fit) %>% arrange(desc(Overall))
rf_im <- importance(rf_fit)
```

```r
new_8 <- rownames(rf_im)[which(abs(rf_im[ , 1]) > 8)]
newdata_8 <- train[ , c(new_8, "growth_2_6")]

rf_8 <- randomForest(growth_2_6 ~ . , data = newdata_8,
                     mtry = floor((ncol(newdata_8) - 1) / 3),
                     ntree = 1000, importance = TRUE)

rf_pred_8 <- predict(rf_8, test1)
RMSE_rf_8 <- sqrt(mean((test1$growth_2_6 - rf_pred_8) ^ 2))
RMSE_rf_8
```

- With 41 predictors and mtry value of 13, RMSE obtained is 1.421445.

```r
rf_8_14 <- randomForest(growth_2_6 ~ . , data = newdata_8,
                     mtry = 14, ntree = 1000, importance = TRUE)

rf_pred_8_14 <- predict(rf_8_14, test1)
RMSE_rf_8_14 <- sqrt(mean((test1$growth_2_6 - rf_pred_8_14) ^ 2))
RMSE_rf_8_14
```

- With 41 predictors and mtry value of 14, RMSE obtained is 1.422817.

```r
rf_8_12 <- randomForest(growth_2_6 ~ . , data = newdata_8,
                        mtry = 12, ntree = 1000, importance = TRUE)

rf_pred_8_12 <- predict(rf_8_12, test1)
RMSE_rf_8_12 <- sqrt(mean((test1$growth_2_6 - rf_pred_8_12) ^ 2))
RMSE_rf_8_12
```

- With 41 predictors and mtry value of 12, RMSE obtained is 1.425424.

```r
rf_im <- importance(rf_w_time)
new_9 <- rownames(rf_im)[which(abs(rf_im[ , 1]) > 9)]
```

```r
newdata_9 <- train[ , c(new_9, "growth_2_6")]

rf_9 <- randomForest(growth_2_6 ~ . , data = newdata_9,
                     mtry = floor((ncol(newdata_9) - 1) / 3),
                     ntree = 1000, importance = TRUE)

rf_pred_9 <- predict(rf_9, test1)
RMSE_rf_9 <- sqrt(mean((test1$growth_2_6 - rf_pred_9) ^ 2))
RMSE_rf_9
```

- With 35 predictors and mtry value of 11, RMSE obtained is 1.423923.

```r
rf_9_12 <- randomForest(growth_2_6 ~ . , data = newdata_9,
                        mtry = 12, ntree = 1000, importance = TRUE)

rf_pred_9_12 <- predict(rf_9_12, test1)
RMSE_rf_9_12 <- sqrt(mean((test1$growth_2_6 - rf_pred_9_12) ^ 2))
RMSE_rf_9_12
```

- With 35 predictors and mtry value of 12, RMSE obtained is 1.422035.

```r
rf_9_10 <- randomForest(growth_2_6 ~ . , data = newdata_9,
                        mtry = 10, ntree = 1000, importance = TRUE)

rf_pred_9_10 <- predict(rf_9_10, test1)
RMSE_rf_9_10 <- sqrt(mean((test1$growth_2_6 - rf_pred_9_10) ^ 2))
RMSE_rf_9_10
```

- With 35 predictors and mtry value of 10, RMSE obtained is 1.421007.

```r
new_10 <- rownames(rf_im)[which(abs(rf_im[ , 1]) > 10)]
newdata_10 <- train[ , c(new_10, "growth_2_6")]

rf_10 <- randomForest(growth_2_6 ~ . , data = newdata_10,
                      mtry = floor((ncol(newdata_10) - 1) / 3),
                      ntree = 1000, importance = TRUE)

rf_pred_10 <- predict(rf_10, test1)
RMSE_rf_10 <- sqrt(mean((test1$growth_2_6 - rf_pred_10) ^ 2))
RMSE_rf_10
```

- With 33 predictors and mtry value of 11, RMSE obtained is 1.423562.

```r
rf_10_12 <- randomForest(growth_2_6 ~ . , data = newdata_10,
                         mtry = 12, ntree = 1000, importance = TRUE)

rf_pred_10_12 <- predict(rf_10_12, test1)
RMSE_rf_10_12 <- sqrt(mean((test1$growth_2_6 - rf_pred_10_12) ^ 2))
RMSE_rf_10_12
```

- With 33 predictors and mtry value of 12, RMSE obtained is 1.412062.

```r
rf_10_10 <- randomForest(growth_2_6 ~ . , data = newdata_10,
                         mtry = 10, ntree = 1000, importance = TRUE)

rf_pred_10_10 <- predict(rf_10_10, test1)
RMSE_rf_10_10 <- sqrt(mean((test1$growth_2_6 - rf_pred_10_10) ^ 2))
```

```r
RMSE_rf_10_10
```

- With 33 predictors and mtry value of 10, RMSE obtained is 1.414207.

```r
new_11 <- rownames(rf_im)[which(abs(rf_im[ , 1]) > 11)]
newdata_11 <- train[ , c(new_11, "growth_2_6")]

rf_11 <- randomForest(growth_2_6 ~ . , data = newdata_11,
                      mtry = floor((ncol(newdata_11) - 1) / 3),
                      ntree = 1000, importance = TRUE)

rf_pred_11 <- predict(rf_11, test1)
RMSE_rf_11 <- sqrt(mean((test1$growth_2_6 - rf_pred_11) ^ 2))
RMSE_rf_11
```

- With 30 predictors and mtry value of 10, RMSE obtained is 1.424145.

```r
# choose our model based on the result above
final_data <- train_left[ , c(new_10, "growth_2_6")]
final_model <- randomForest(growth_2_6 ~ . , data = final_data,
                            mtry = 12, ntree = 1000, importance = TRUE)
newtest <- test[ , new_10]
pred_test <- predict(final_model, newdata = newtest)
result_w_days <- data.frame(test$id, pred_test)
colnames(result_w_days) <- c("id", "growth_2_6")
write.csv(result_w_days, "result_w_days_10_10.csv", row.names = FALSE)
```