Final Report

Team Name: GPA 4.0
Member: Violet(Bingcui) Guo, Jonathan(Yiran) Liu

I. Introduction

A video's early growth pattern can indicate the eventual success of the video and also the overall health of the channel. The goal of the project was to predict the percentage change in views on a video between the second and sixth hour since its publishing with features of thumbnail image, video title and some other features like the duration of the video.

II. Methodology

A. Data Preprocessing

After we loaded the data into R, we first checked for missing values. Since there were no missing values found in the dataset, we didn't perform any missing values handling techniques. Carefully reviewing the Feature_Desriptions document, we found "PublishedDate" particularly valuable for the rich information this predictor provides because from common sense, the published weekday, month, and the time of a day that the video is published would affect the number of views on the video. With a format of Month/Day/Year Hour:Minute, we decided to dissect the "PublishedDate" into five distinct predictors with the help of function as.Date from the library(lubridate) (We've asked for permission to use the package via email) (Vitalie, 2020). The five predictors are, "years", "months", "dates", "weekdays" and "myduration". While the first four are self-explanatory, "myduration" is the amount of minutes past after 0am (So 1:00 corresponds to 60 and 23:30 corresponds to 60*23+30= 1410)

Removing highly correlated predictors is essential for later performing Random Forest to conduct feature selection. When there is a high correlation between two predictors, any of the two can be used as the predictor. But if one of them is used in the model, the importance of the other one is reduced since the impurity they can remove is removed by the selected predictor. This became a problem when we took the most important predictors to build the model.

To remove highly correlated predictors, we used the function findCorrelation function from library(mlbench) (Friedrich and Evgenia, 2012) and function vsetdiff from library(vecsets)(Carl. W., 2018) (We've asked for permission to use the packages via email). Firstly, we extracted the numerical predictors in the training data sets, and then we used the cor function to get the correlation matrix of all the pairs. And then, I used findCorrelation function to find pairs with a correlation coefficient of over 0.7 and used the function vsetdiff to remove one predictor of each pair.

We decided to set the cutoff to 0.7 because cutoff was a parameter to decide what we deem as high enough correlation to toss out certain predictors. In other words, we would only keep predictors that have correlations below the cutoff value. We set the cutoff to 0.7 and that is because statisticians in general (Diana and Phoebe, 2014), consider a correlation of 0.7 as highly correlated. Furthermore, scaling the predictors was not a concern since we have decided to only use Bagging and Random Forest to build the model and tree-based models do not require feature scaling.

B. Feature Selection

Due to the large number of predictor variables in the given data set, the presence of non-informative variables can add uncertainty to the predictions and reduce the overall effectiveness of the model. Thus, feature selection became one of our most important tasks. We came up with two strategies for feature selection. Our first approach is Lasso. The reason that we preferred Lasso over Ridge is Lasso makes unimportant predictors to have zero coefficients, which resulting in an easier choice of identifying important predictors as compare to Ridge, which only shrinks the coefficients of unimportant predictors to a very small value. And our second approach is building a model with Random Forest in which all predictors from the dataset are included, and then we use the importance function to acquire the most important predictors. The reason why we used random forest to do feature selection is that the tree-based strategies used by random forests naturally rank by how well they improve the purity of the node. With the help of the histogram of the frequency of feature importance plot below(Figure 1), we know that we want the predictors that are of high importance and we don't want that many predictors. So from the plot, we could see that the predictors that are of value are in the tail of the distribution. So we dropped the predictors in the first 2 rectangles and used the predictors in the tail by setting the benchmark of 8 and we got 30 predictors. We ran a base default random forest with these 30 predictors by using 70% training data and got a RMSE result on the remaining 30% data of 1.47996. And if we increase the number of predictors to 31, 32, ...36, we get worse results of 1.48205, 1.48403, … 1.50005. So with more predictors, the model gets worse in terms of interpretability. So we chose 30 predictors as the predictors for the final model.

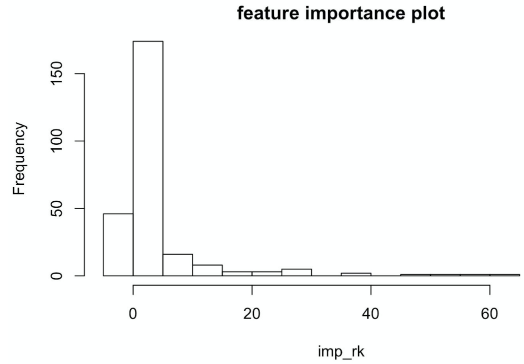

Figure 1

With the above two approaches, we had one group of features from Lasso and another group of features from Random Forest. We decided to use these two groups of features to build two models that depend on them respectively. And then, we have decided to compare oob for the . The details will be explained in the section of Model Selection Process.

C.Model Selection Process

We had separated the dataset from the training.csv into our own training data (70 percent) and testing data (30 percent) at the very beginning. In this way, we could build models from our training data solely to evaluate their performances and compare them with each other. This evaluation was based on a RMSE calculating function we wrote consulting the formula given on Kaggle. By doing so, not only we were able to get a general sense of our models' performance prior to submitting on Kaggle, but also we could choose the best model. Since the models we built were only based on 70 percent of the date from the original dataset in training.csv, after selecting our best model, we then used the whole dataset to build the same model, which in return, did give us a better score after submitting on Kaggle. It only took 7 submissions to get a score of 1.37665 on the public board and we credited this efficiency to our evaluation approach. We have two groups of predictors from Feature Selection; the predictors selected by Lasso and the predictors selected by Random Forest Importance function. We chose the best group by building two different Random Forest models and then compare the Out-of-bag (OOB) error. The following was the result (Figure 2 and Figure 3). Why we build random forests as a base model is because it utilizes an ensemble method which adds more diversity to the trees and works well in high-dimensional data. And we used OOB error allows us to estimate the prediction performance improvement by evaluating predictions on those observations which were not used in the building of the next base learner.

Model built with predictors selected by Lasso

	RMSE <dbl>	Rsquared <dbl>	mtry <dbl>
1	1.920184	0.4723282	2
2	1.633841	0.6179697	52
3	1.646280	0.6121305	103

Figure 2

Model built with predictors selected by Random Forest Importance

	RMSE <dbl>	Rsquared <dbl>	mtry <dbl>
1	1.559503	0.6480864	2
2	1.494743	0.6767067	16
3	1.486695	0.6801788	30

Figure 3

So as we can see, the models built with predictors selected by Random Forest Importance gave a better RMSE. So we had decided to use the predictors selected by Random Forest Importance. In order to achieve the best Random Forest performance possible, the tuning of hyperparameters, namely, "mtry" (Number of variables randomly sampled as candidates at each split) and "ntree" (Number of trees to grow), was necessary. With the plot of squared error vs ntrees below(Figure4 and Figure5), it was obvious that the error (square RMSE) was the lowest when "ntree"=1100. Thus, we set the parameter "ntree" to be 1100 in our final model. To verify this decision, we had tried to build Random Forest models with "ntree"'s equal to "1100","1500" and "2000". And evaluating the model on the test dataset, we get the results of 1.459922, 1.461391, and 1.461919 respectively. The reason that these test MSE's were lower than our public score was that we only used our own training data to train the model, which only contained 70 percent of the whole dataset in training.csv.

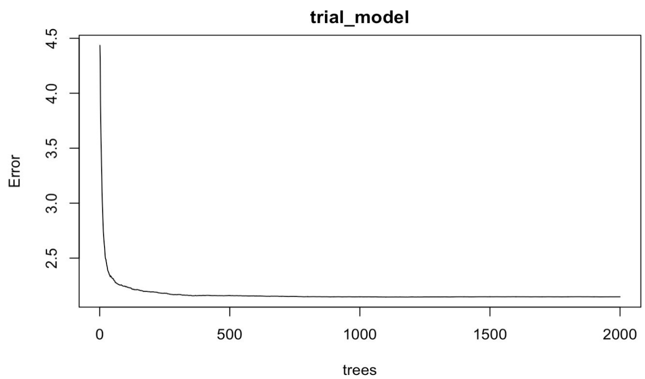

Figure4

Figure5

From the following plot of Out of bag RMSE vs number of predictors (Figure6), we can see when mtry is 30, we get the lowest MSE.

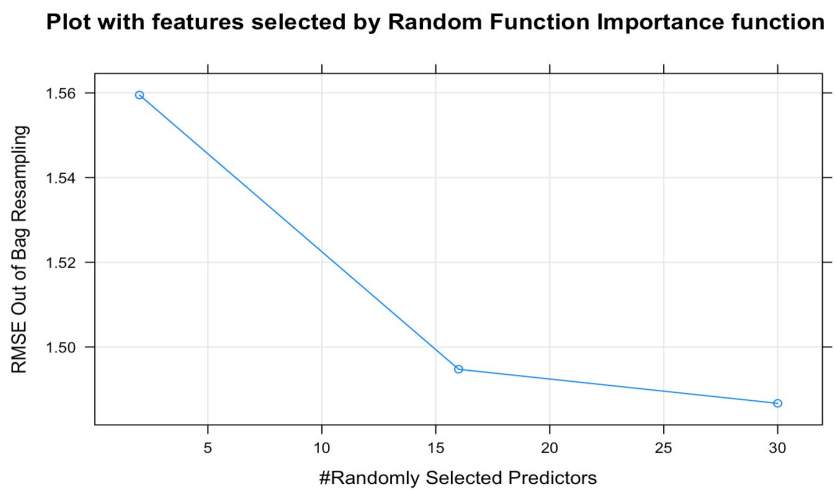

Figure6

Besides Random forest, we have also used Ridge Regression to build a model because ridge allows us to “shrink” unimportant coefficients, which makes it work well on new datasets and use complex models while avoiding overfitting at the same time. By using cross-validation, we found the best lamda. The model built by Ridge Regression with the best lamda we found did not give us a good performance at all. The RMSE is 1.70454, which is less than the RMSE we received from using Random Forest. After we had made the decision of using the Bagging model with “mtry” equals to 30 and “ntree” equals to 1100 as our final model. We trained the model with the whole dataset from training.csv and used this model to predict. We got 1.45467 by using RMSE on our laptop by using 70% data and got a RMSE of 1.37665 by using all the data on Kaggle public board and 1.39300 on private board.

III. Result

Our final model is built by Bagging with “mtry” equals to 30 and “ntree” equals to 1100 and we got a RMSE of 1.37665 on the Kaggle public board, which is a very decent score.

IV. Conclusion

We received a score of 1.37665 on the Kaggle public board and 1.39300 on the Kaggle private board, which is very similar. Thus, our model has a good interpretability. We credit this good performance to our tactic of using creative preprocessing method, creating our own testing dataset, so we can evaluate our models performance prior to submission and by using Out-of-bag error and Cross-Validation, we improved our model and avoided over-fitting. Ideas on further improving our model includes tuning more parameters such as maxnodes and nodesize and using some more advanced feature selection methods not seen in class, such as recursive feature elimination. We hope to learn more and improve our data science skills in the future.

V. Statement of Contribution

Jonathan: Data Preprocessing, Data Modeling, Report Writing
Violet: Data Preprocessing, Feature Selection, Data Modeling, Report Writing

VI. Reference

Carl, Wi., 2018. *Package 'vecsets'*. Retrieved 2018 (https://cran.r-project.org/web/packages/vecsets/vecsets.pdf)

Diana, M. and Phoebe B. 2014. *Scatterplots and Correlation*. Retrieved 2014 (https://www.westga.edu/academics/research/vrc/assets/docs/scatterplots_and_correlation_notes.pdf).

Evgenia, D and Friedrich, L. 2012. *Package 'mlbench'*. Retrieved 2012 (https://cran.r-project.org/web/packages/mlbench/mlbench.pdf)

Vitalie, S., 2020. *Package 'lubridate'*. Retrieved 2020 (https://cran.r-project.org/web/packages/lubridate/lubridate.pdf).

VII. R Appendix

````
```{r}
##read in data
data <- read.csv("training.csv")

```
````

````
```{r}
##load library
library(lubridate)
library(caret)
```
````

````
```{r}
dim(data)
```
````

```
```{r}
## data transformation
years1 <- c()
months1 <- c()
days1 <- c()
hours1 <- c()
minutes1 <- c()
weekday1<- c()
for (i in 1:7242){
  mydate <- strptime(data[,"PublishedDate"][i] , format = "%m/%d/%Y%H:%M")
  years1 <- c(years1, year(mydate))
  months1 <- c(months1, month(mydate))
  days1 <- c(days1, day(mydate))
  hours1 <- c(hours1, hour(mydate))
  minutes1 <- c(minutes1, minute(mydate))
  weekday1 <- c(weekday1, weekdays(mydate))
}
```


```{r}
##remove id and publsihed date
data$myduration <- 60*hours1 + minutes1
data <- data[, -c(1,2)]
```

```{r}
##add transformed columns to the data
data$years  <- years1
data$months <- months1
data$dates <- days1
data$weekdays <- weekday1
data$weekdays <- as.factor(data$weekdays)
```
```

````
```{r}
## train_test split
size <- dim(data)[1]
train_idx <- sample(1:size, floor(size*0.7))
train <- data[train_idx,]
test <- data[-train_idx,]


```

```{r}
##remove highly correlated predictors
# Splitting numeric and categorical variables

numeric_idx <-c(3:247, 261:262)
categorical_idx <- c(248:259)
total_colnames <- colnames(data)

set.seed(1)
library(mlbench)
library(caret)
library(vecsets)
# calculate correlation matrix
correlationMatrix <- cor(train[,numeric_idx],use="complete.obs")
is.na(train)

train<-train[,colSums(is.na(train))==0]


#remove columns with std=0
predictors <- names(which(is.na(correlationMatrix[1,])))
predictors # all numeric
idx <- match(predictors,total_colnames)
numeric_idx2 <- numeric_idx[!numeric_idx %in% idx]
numeric_idx2

````
````

```
# find attributes that are highly corrected
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.7)
numeric_highly_cor <- vsetdiff(numeric_idx2, highlyCorrelated)
length(numeric_highly_cor)
numeric_highly_cor

##name of all the highly correlated data
total_colnames[numeric_highly_cor]
```

```{r}
## lasso feature selection
lasso.mod <- glmnet(train.mat, train$growth_2_6, family = "gaussian", alpha = 1, lambda=grid,
                     standardize = TRUE)
```

```{r}
## cv lasso
cv.lasso <- cv.glmnet(train.mat, train$growth_2_6, alpha=1, lambda=grid, standardize=TRUE, nfolds=10)
```

```{r}
cv.lasso$lambda.min
```

```{r}
## see important coefficients
 d <- predict(lasso.mod, s=cv.lasso$lambda.min, type="coefficients")
```

````
```{r}
##see important features
sig <- dimnames(d)[[1]][which(d!=0)]
sig1 <- sig[-1]
sig
```

```{r}
## look at lasso prediction error by using linear regression
## and we know that the number of predictors that lasso selects out are too much
## and the lasso prediction is so abd
predict(lasso.mod, s = cv.lasso$lambda.min, type="coefficients")
testx <- model.matrix(growth_2_6~.,test)
testy <- test$growth_2_6
lasso.pred = predict(lasso.mod, newx= testx, s=cv.lasso$lambda.1se)
lasso.err = mean((testy - lasso.pred)^2)
```

```{r}
lasso.err
```

```{r}
##select the predictors after removing all the highly correlated data
data <- data[, -c(total_colnames[numeric_idxe])]
```

```{r}
## do train_test split again by using the new data
size <- dim(data)[1]
train_idx <- sample(1:size, floor(size*0.7))
train <- data[train_idx,]
test <- data[-train_idx,]
```
````

```
```{r}
##feature selection rf
rf_feature_select <- randomForest(growth_2_6~., data=train, mtry=ncol(train)-1, n.trees=1000, importance=T)
```

```{r}
## look at importance features
imp_rk <- sort(importance(rf_feature_select)[, 1],decreasing=TRUE)
```

```{r}
##look at the most importance features
varImpPlot(rf_feature_select, pch=19, cex=0.6)
```

```{r}
## generate variable importance histagram plot to see the distribution
## of the importance score and the frequency the varaibles with that score appear
var_index <- importance(rf_all)[,1]
hist(var_index,density = FALSE,main = "feature importance plot",xlab = "feature importance value", xlim = c(-50,150),breaks = 20)
```


```{r}
## find the optimal number of trees
library(randomForest)
trial_model <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342, data=train, ntree=2000, mtry=30)
plot(trial_model)
plot(trial_model,ylim=c(2.14,2.15),xlim=c(900,2100))

```
```

```{r}
## test out the optimal number of predictors
trial_model1 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641, data=train, ntree=2000, mtry=31)
trial_model1$results

trial_model2 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641+cnn_30, data=train, ntree=2000, mtry=32)
trial_model2$results


trial_model3 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641+cnn_30+week, data=train, ntree=2000, mtry=33)
trial_model3$results

trial_model4 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641+cnn_30+week+year, data=train, ntree=2000, mtry=34)
trial_model4$results

trial_model5 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641+cnn_30+week+year, data=train, ntree=2000, mtry=35)
trial_model5$results

trial_model6 <- randomForest(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342+hog_641+cnn_30+week+year, data=train, ntree=2000, mtry=36)
trial_model6$results
```

```r
train_control <- trainControl(
                          method = "oob",

                         savePredictions = TRUE
                          )

## find OOB error to compare rf and bagging
treefit1<- train(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342,
                data = train, method = 'rf',
                trControl = train_control,ntree=2000)

plot(treefit1,main="Plot with features selected by Random Function Importance function")

treefit2<- train(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342,
                data = train, method = 'rf',
                trControl = train_control,ntree=1100)

plot(treefit2,main="Plot with features selected by Random Function Importance function")

treefit3<- train(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342,
                data = train, method = 'rf',
```

````
                 data = train, method = 'rf',
                 trControl = train_control,ntree=1500)
## plot the resluts using differnet ntrees
par(mfrow=c(3,1))
plot(treefit1,main="Plot with features selected by Random Function Importance function ntree=2000",xlab="mtry")
plot(treefit2,main="Plot with features selected by Random Function Importance function ntree=1000",xlab="mtry")
plot(treefit3,main="Plot with features selected by Random Function Importance function ntree=1500",xlab="mtry")
treefit1$results
treefit2$results
treefit3$results


treefit<- train(growth_2_6~cnn_10+cnn_17+cnn_86+cnn_89+cnn_25+cnn_12+views_2_hours+cnn_68+punc_num_..21+cnn_88+punc_num_..28+Duration+num_uppercase_chars+num_words+hog_454+num_digit_chars+num_chars+punc_num_..1+sd_blue+hog_453+mean_red+sd_green+num_uppercase_words+hog_859+hog_651+punc_num_..11+hog_359+doc2vec_4+hog_668+punc_num_..14+hog_669+doc2vec_10+num_stopwords+hog_182+hog_295+hog_132+hog_177+hog_702+hog_40+hog_649+hog_738+hog_62+hog_716+doc2vec_2+doc2vec_8+hog_832+hog_657+hog_829+hog_791+hog_336+hog_94+punc_num_..15+hog_674+doc2vec_9+hog_350+doc2vec_11+hog_492+hog_452+pct_nonzero_pixels+hog_746+hog_60+hog_774+hog_844+hog_724+hog_640+doc2vec_15+hog_1+punc_num_..8+hog_341+hog_342+hog_78+hog_855+hog_125+doc2vec_3+hog_677+hog_782+hog_316+hog_849+hog_705+hog_815+hog_378+hog_61+hog_195+hog_116+cnn_39+punc_num_..3+punc_num_..7+hog_641+hog_676+hog_279+hog_523+hog_665+punc_num_..2+hog_133+hog_856+hog_655+max_red+punc_num_..16+punc_num_..20+doc2vec_13+Num_Subscribers_Base_low+avg_growth_low_mid+count_vids_low,
                 data = train, method = 'rf',
                 trControl = train_control,ntree=2000)

plot(treefit,main="Plot with features selected by Lasso")
treefit$results

```

````
````

```{r}
## funciton to get RMSE
RMSE <- function(y1,y2){
  rmse <- sqrt(mean((y1-y2)^2))
  return(rmse)
}
```

```{r}
pred_test <- predict(trial_model, test)
pred_test1 <- predict(treefit1, test)
pred_test2 <- predict(treefit2, test)
pred_test3 <- predict(treefit3, test)
```

```{r}
##cross validation to see the result
RMSE(test$growth_2_6, pred_test2)
RMSE(test$growth_2_6, pred_test1)
RMSE(test$growth_2_6, pred_test2)
RMSE(test$growth_2_6, pred_test3)

```

```{r}
##read in test data
test_id <- read.csv("test.csv")
```

```{r}
## read in test data
test_data <- read.csv("test.csv")
```
```{r}
## transform test data
years3 <- c()
months3 <- c()
```

```
months3 <- c()
days3 <- c()
hours3 <- c()
minutes3 <- c()
weekday3<- c()
for (i in 1:3105){
  mydate <- strptime(test_data[,"PublishedDate"][i] , format = "%m/%d/%Y%H:%M")
  years3 <- c(years3, year(mydate))
  months3 <- c(months3, month(mydate))
  days3 <- c(days3, day(mydate))
  hours3 <- c(hours3, hour(mydate))
  minutes3 <- c(minutes3, minute(mydate))
  weekday3 <- c(weekday3, weekdays(mydate))
}
```

```{r}
test_data$years  <- years3
test_data$months <- months3
test_data$dates <- days3
test_data$weekdays <- weekday3
test_data$myduration <- hours3 * 60 + minutes3
```

```{r}
## get our output
df1<-data.frame(test_id[,1],pred_labal2)
colnames(df1) <- c("id", "growth_2_6")

write.csv(df1,"submission_bagging_final2.csv", row.names = FALSE)


# ridge regression trial

library(glmnet)

i.exp <- seq(10, -2, length = 100)
grid <- 10^i.exp

69:1 (Top Level) ÷                                    R Markdown ÷
```

````
i.exp <- seq(10, -2, length = 100)
grid <- 10^i.exp

x <- model.matrix(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342,train)[,-which(names(train) %in% "growth_2_6")]
y <- train$growth_2_6

x1 <- model.matrix(growth_2_6~avg_growth_low_mid+cnn_10+Num_Views_Base_mid_high+avg_growth_low+cnn_86+cnn_89+cnn_12+cnn_17+Num_Subscribers_Base_mid_high+Num_Subscribers_Base_low_mid+views_2_hours+cnn_25+myduration+count_vids_low_mid+avg_growth_mid_high+cnn_88+cnn_68+count_vids_mid_high+punc_num_..28+num_uppercase_chars+Duration+num_words+cnn_19+num_digit_chars+punc_num_..21+num_chars+punc_num_..1+months+mean_green+hog_342,test)[,-which(names(test) %in% "growth_2_6")]


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


cv.output1 <- cv.glmnet(x, y, family = "gaussian", alpha = 1,
lambda = grid, standardize = TRUE,
nfolds = 10)

best.lambda.cv1 <- cv.output1$lambda.1se

Ridge_result<-predict(lasso.mod, s = best.lambda.cv1,
newx=x1)


RMSE(test$growth_2_6,Ridge_result)

#RMSE is 1.70454
```
````
````