Predicting the Growth Rate of Youtube Videos

Team: COVARIANCE-19 (Yeryeong Choe, Cassandra Tai, Richard Yim)

14 December 2020 – Department of Statistics, UCLA

1 Introduction

One of a content creator's goals is to know how fast a video will grow in order to determine how much revenue they will earn, which can be indicated by a video's early growth pattern. In particular, this project aims to predict the percentage change increase in views on a video between the second and sixth hour since it was publishd to more broadly estimate the overall success of the video and channel.

2 Data Preprocessing

There are no missing values from the provided data, so no imputation was required. The data contains 7242 videos (observations) and 258 predictors categorized by four main features: thumbnail image (pixel characteristics), video title (number of character and words), channel (number of subscribers), and other (video duration, number of views, and the time stamp). The training data also includes a response variable `growth_2_6` that is bounded between 0 and 10 to represent how many times the video's views have increased by from the second to sixth hour since publishment.

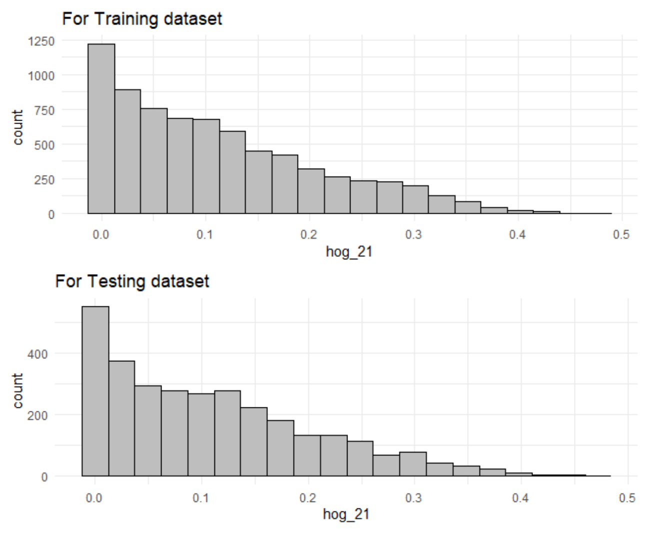

Figure 1: Histogram of non-normal, skewed numeric values (similar for many of the predictors) for 'hog 21' on oriented gradients for a particular gradient in the thumbnail of a video.

For preliminary observations, boxplots, histograms, and statistical summaries were used to observe quantiles and overall numeric ranges for the 258 predictors for both training and testing datasets to generate an understanding on how the numeric values of each variable for each dataset could affect a range of possible models. For example, in applying a tree or random forest on the training dataset, the testing set for prediction would need to have similar range of values between the training and testing sets for transferrable predictions such as in Figure 1. One important predictor required transformation for model interpretability: `publishedDate`, was in factor form and included the time and date marking the publishing date of the video. More views are expected happen in the day, so numerical values would be required to encode such significance in which videos published in the day would experience more views than videos published late at night. Three numerical predictors were consequently constructed from this variable.

First, a `Time` variable, which reflects how many minutes past midnight PST the video was uploaded, was created. Second, a `daysSince0411` variable that represents how many days past the day the earliest video in the datasets was published was created. Finally, a `dayOfWeek` variable was constructed by taking `daysSince0411` mod 7, marking numeric values 0 through 6 and days of the week (actual day representation was not considered, only the numeric classification). The day of the week and time of day a video is posted are considered to intuitively be important predictors in the early growth of a video, as mentioned earlier. Aside from engineering this important feature, a few predictors had constant values of 0 across all observations, such as `cnn_35`, and were consequently removed from consideration as they provided no variation in observations.

3 Feature Selection and Statistical Modelling

3.1 Linear Regression and Leverage Point Analysis

Linear models were initially considered. Since there were numerous predictors, only the least correlated variables among each feature were chosen. This was done heuristically to subset as few variables as possible where regression would be feasible to minimize inflation of the estimated coefficient variance. Using leverage point analysis, a conventional statistical method to filter outliers, few points of high leverage were observed, which are observations with hat-values that lie outside the conventional leverage boundary of $\frac{4}{n} = \frac{4}{7242} \approx 0.00055$, where $n = 7242$ is the number of observations. However, these high leverage points were "good" as their studentized residuals were bounded in the interval [-3,3], indicating that these outlier observations positively contributed to reducing linear model errors. A QQ-plot in Figure 2 (left) indicates that this method is valid, and the influence plot in Figure 2 (right) shows the distribution of hat-values with respected to studentized residuals. With this in mind, all observations were maintained for other models.

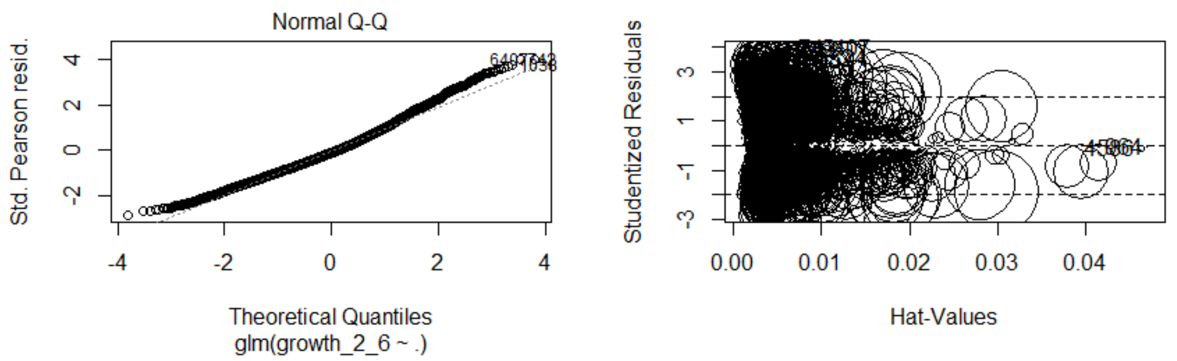

Figure 2: QQ-plot of theoretical and empirical residuals (left). Influence plot given hat-value and studentized residuals (right).

Other more complex regression models were attempted: regularized regression, both ridge and lasso using best λ from cross-validation; partial least squares regression and principal component regression using best number of components from cross-validation simulations. In general, these models were consistently scoring high in RMSE error on the validation (public leaderboard) dataset. This is mostly explained by how our predictors were often found to be non-normal, as seen in Figure 1, and how least squares regression is a projective method that can otherwise remove information important to regression. Furthermore, since the dataset had important features that are non-linearly related, linear models performed poorly.

3.2 Random Forest and Predictor Importance

In comparison, non-parametric models that can retain high dimensionality and simultaneously reduce bias and variance were used: bagging and random forest. Decision trees are simple and useful models for interpretation, but they are highly dependent on the data trained on, consequently having high variance. By averaging a number of decision trees into one single model, bagging and random forests can reduce this variance while maintaining low bias, and thus are powerful models. While these models require a lot of computation with more trees and lose interpretability, statistical inference is not the aim of the project, but accurate predictions. Furthermore, these models are both robust to outliers because high leverage values are categorized to a node rather than a tight ordinary least square regression fit as discussed previously. The bootstrap ingrained in the random forest function also introduces randomization in our observations to train our model on, and hence reduces variance and develops a stronger generalization power.

For feature selection, because correlation subset is relative to linear regression models, only the important variables from the random forest model indicated by `%IncMSE` scores were used by considering half of all predictors at each node for features selection. Various thresholds of `%IncMSE` were used to determine heuristically which predictors to use for the bagging and random forest models; these were deemed the predictors that would increase MSE the most if removed from the model, and consequently are the most important to retain. Figure 3(left) reflects the predictors chosen for a threshold of 0.008, which we used in one of our final models. This threshold value was chosen because this particular set of predictors produced the best out-of-bag (OOB) error score from the random forest model.

From this set of around 30 predictors, multicollinearity was also observed to remove any strongly linearly dependent (correlated) predictors. From the heatmap in figure 3 (right) representing a subset of these predictors, predictors `cnn_12`, `cnn_17`, and `cnn_86` are highly correlated; predictors `cnn_17` and `cnn_86` were then removed to prevent overfitting on predictors with similar characteristics and identify the truly strong predictors.

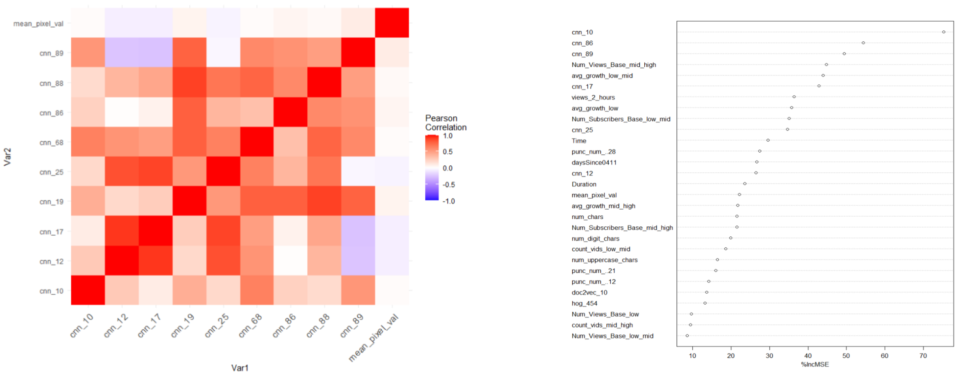

Figure 3: Correlation heatmap of selected sampled predictors from best random forest subset (left). Importance rankings of predictors; higher %IncMSE score corresponds to higher importance in random forest (right).

Initially, predictors based on a threshold of 0.01, excluding the ones removed due to multicollinearity, were considered valuable and bagging was performed with 500 trees to reduce variance. While bagging was though to perform with the most precision, as it considers all predictors at each split, there was also the concern that the model contains many correlated trees and effectively would not reduce variance. To mitigate this, a random forest was trained, which also randomizes what potential predictors can be used at each split, and hence improve accuracy. And while random forest is generally robust to overfitting due to bootstrapping observations, randomly selecting the given parameter `mtry` of predictors to consider at each node, it is still prone to overfitting. With this consideration, hyperparameters were adjusted: decreasing `mtry` and `maxnodes`, and increasing `ntree` and `nodesize` to develop shorter trees and generalize the model by averaging out more trees. This produced a marginal increase in to the OOB error score, but this bias was understood to be compensated by more consistent errors and lower variance on validation and testing data.

4 Results

To evaluate models on Kaggle, the root mean squared error (RMSE) metric, measuring the distance between predicted growth percentage and the true growth percentage, was used (lower RMSE is better). For the random forest and bagging models in R, because both are developed from averaging a number of trees, the square root OOB error was used for comparison. The selected bagging and random forest models achieved a OOB CV error of 1.43556 and 1.464897, which translated to a public score of 1.37148 and 1.40449 on Kaggle respectively, positioning the team to 3rd on the public leaderboard.

5 Conclusion

On the private scoreboard, the bagging model achieved a score of 1.38219 and the random forest model 1.41259. Both models performed slightly worse on the private scoreboard compared to the public, suggesting that the models did have some slight variance. However, the random forest model error did not increase as much, potentially suggesting that adjustments in hyperparameters had good influence in predictive accuracy. Because predictors were selected for bagging and random forest based on first, importance %IncMSE and second, collinearity, further improvements could be made to the model. For example, because collinearity undermines the true strength of a predictor, the usage of statistical methods such asPCA to reduce dimensionality and multicollinearity could be applied first before ranking predictor importance. Furthermore, understanding the data better–including the HOG and CNN feature effects–and taking an alternative approach for feature selection using domain knowledge of common trends among YouTube channel would allow us to truly know which particular variables contribute to a video's view count. But overall, the models worked very well as the final RMSE positioned the team to rank 1 on the private leaderboard.

6 Appendix (R Code):

6.1 Initial Data Exploration and Analysis of Numeric Quantities

```r
# ============================== #
#  LOADING DATA
# ============================== #
training <- read.csv("../CSVs/training.csv", header=TRUE)
testing <- read.csv("../CSVs/test.csv", header=TRUE)

# ============================== #
#  STATISTICAL SUMMARIES
# ============================== #
#first check for any NAs-- None!
sum(is.na(training))
sum(is.na(test))

begin_index <- 2 #change this!
end_index <- 10 #change this!
summary(training[,begin_index:end_index])
summary(training$growth_2_6)

# ============================== #
#  BOXPLOTS
# ============================== #
begin_index <- 2 #change this!
end_index <- 10 #change this!
attach(training)
for (feat in names(training)[begin_index:end_index])
#for (feat in vars_to_keep)
{
  if (feat != 'growth_2_6')
    #wasn't sure whether to floor or round growth_2_6
  {
    p = ggplot(data = training, aes_string(growth_2_6, y=feat, group = round(growth_2_6))) +
  geom_boxplot()
    print(p)
  }

}

# ============================== #
#  HISTOGRAMS
# ============================== #
begin_index <- 4 # change this!
end_index <- 12 # change this!
library(ggplot2)
for (var in names(training)[begin_index:end_index]) {
  # so we don't make 200+ graphs
  curr_plot = ggplot(training, aes_string(x=var))
  if(is.factor(training[,var]))  {
    curr_plot = curr_plot + geom_bar() # discrete

  }
  else  {
    curr_plot = curr_plot + geom_histogram(bins=20) # continuous
  }
  print(curr_plot)
}
# For our report:
library(gridExtra)
g1 <- ggplot(training, aes(x=hog_21))+ geom_histogram(bins=20, color = "black", fill = "gray") +
  theme_minimal() + labs(title = "For Training dataset")
g2 <- ggplot(test, aes(x=hog_21))+ geom_histogram(bins=20, color = "black", fill = "gray") +
  theme_minimal() + labs(title = "For Testing dataset")
grid.arrange(g1,g2, nrow = 2)

# ============================== #
#  CORRELATION HEAT MAP
# ============================== #
begin_index <- 3 #change this!
end_index <- 10 #change this!
#cor_mtx = round(cor(training[,vars_to_keep]), 2) #so we don't get a hugeeeee table
cor_mtx = round(cor(training[,begin_index:end_index]), 2)
#reshape it
melted_cor_mtx <- melt(cor_mtx)

#draw the heatmap
cor_heatmap = ggplot(data = melted_cor_mtx, aes(x=Var1, y=Var2, fill=value)) + geom_tile()
cor_heatmap = cor_heatmap +
        scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                             midpoint = 0, limit = c(-1,1),
                             space = "Lab", name="Pearson\nCorrelation") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, vjust = 1, size = 12, hjust = 1))
cor_heatmap
```

6.2 Leverage Analysis

```
# ============================== #
#  LIBRARIES AND PATH
# ============================== #
set.seed(42)
library(car)
library(MASS)

# ============================== #
#  LOADING DATA
# ============================== #
training <- read.csv("../CSVs/training.csv", header=TRUE)
testing <- read.csv("../CSVs/test.csv", header=TRUE)
cat(dim(training), dim(testing))
youtube_data <- data.frame(training)
attach(youtube_data)

# ============================== #
#  VARIABLE SUBSET
# ============================== #
# TRY EITHER THE
# BEST_SUBSET_COR SUBSET # OR
# BEST_SUBSET_RF SUBSET

# **************************************** #
#  FIT VARIABLES TO LINEAR MODEL (cor)
# ***************************************** #
glm.fit <- glm(growth_2_6~., data=youtube_data[,best_subset_cor])
# studentized pearson residual Q-Q plot
plot(glm.fit,2)
influencePlot(glm.fit)
# default hat tolerance for more than 1,000 data points *heuristic*
hat_tolerance <- 4/dim(youtube_data)[2]
hat_tolerance
# cutoff at studentized residuals and hat values ranges
unleveraged_rows <- (studres(glm.fit)<2.5) & (hatvalues(glm.fit) < 0.015)
```

6.3 Correlation Variable Selection

```
# ============================== #
#  SETTING UP THE WORKSPACE
# ============================== #
#  LIBRARIES AND PATH
library(ggplot2)
library(boot)
library(caret)
library(leaps)

#  LOADING DATA
setwd('~/101C/KaggleFinal/')
training <- read.csv("training.csv", header=TRUE)
testing <- read.csv("test.csv", header=TRUE)
cat("Training Dimensions:",dim(training),
    "Testing Dimensions",dim(testing))
# remove the ID number
youtube_data <- data.frame(training[,-1])
attach(youtube_data)

# ============================== #
#  VARIABLE SELECTION
# ============================== #
# STATISTICAL SUMMARIES
summary(youtube_data)

# ***************************** #
# THUMBNAIL IMAGE FEATURES
# ***************************** #
# histogram of gradient features
hog_tau <- 0.00005
hog_vars <- which(grepl("hog", names(youtube_data)))
hog_important <- hog_vars[which(abs(cor(
  youtube_data[,hog_vars], youtube_data[,hog_vars])) < hog_tau
      , arr.ind=TRUE)[,"col"]]
hog_important

# most frequent and important cnn extracted features
cnn_tau <- 0.4
cnn_vars <- which(grepl("cnn", names(youtube_data)))
cnn_freq <- cnn_vars[which(apply(youtube_data[,cnn_vars]==0, FUN=sum, MARGIN=2) < 7000)]
cnn_important <- cnn_freq[which(abs(cor(
  youtube_data[,cnn_freq], youtube_data[,cnn_freq])) < cnn_tau
```

```r
      , arr.ind=TRUE)[,"col"]]
cnn_important

# pixel and rgb value features
pixel_tau <- 0.1
pixel_vars <- which(grepl(c("red|blue|green|pixel|edge"), names(youtube_data)))
pixel_vars <- pixel_vars[!grepl("min|max",names(youtube_data[,pixel_vars]))]
pixel_important <- pixel_vars[unique(which(abs(cor(
  youtube_data[,pixel_vars], youtube_data[,pixel_vars])) < pixel_tau
      , arr.ind=TRUE)[,1])]
pixel_important

# ***************************** #
# VIDEO TITLE FEATURES
# ***************************** #
# pixel and rgb value features
nlp_tau <- 0.001
nlp_vars <- which(grepl(c("num|doc"), names(youtube_data)))
nlp_freq <- nlp_vars[which(apply(youtube_data[,nlp_vars]==0, FUN=sum, MARGIN=2) < 7000)]
nlp_important <- nlp_freq[unique(which(abs(cor(
  youtube_data[,nlp_freq], youtube_data[,nlp_freq])) < nlp_tau
      , arr.ind=TRUE)[,1])]
nlp_important

# ***************************** #
# CHANNEL FEATURES
# ***************************** #
channel_tau <- 0.01
channel_vars <- which(grepl(c("count|avg_g|Num"), names(youtube_data)))
channel_important <- channel_vars[unique(which(abs(
  cor(youtube_data[,channel_vars], youtube_data[,channel_vars])) < channel_tau
      , arr.ind=TRUE)[,1])]
channel_important

# ***************************** #
# FINAL VARIABLES INDEX SET
# ***************************** #
important_families <- unique(c(hog_important, cnn_important, pixel_important,
                               nlp_important, channel_important))
important_families
families_tau <- 0.01
important_vars <- important_families[unique(which(abs(cor(youtube_data[,
        important_families], youtube_data[,important_families])) < families_tau
      , arr.ind=TRUE)[,1])]
length(important_vars)

# ============================= #
#  SUBSET SELECTION
# ============================= #
# using important_vars from the variable selection pipeline
regfit.full <- regsubsets(growth_2_6~.,
                          data=youtube_data[,important_vars],
                          nvmax = length(important_vars))
reg.summary <- summary(regfit.full)
# which subset performed the best
best.subset <- which(min(reg.summary$rss)==reg.summary$rss)
length(important_vars)
```

```r
best.subset
best_subset_cor <- names(reg.summary$which[best.subset,
          ])[reg.summary$which[best.subset,]][-1]

best_subset_cor <- c("hog_828"        ,"cnn_12"        ,"cnn_17"
                     ,"cnn_19"        ,"cnn_25"        ,"cnn_88"
                     ,"cnn_89"        ,"doc2vec_16"    ,"avg_growth_low_mid"
                     ,"Num_Views_Base_low"   ,"count_vids_low", "hog_514"
                     , "hog_747"      , "cnn_86"       , "pct_nonzero_pixels"
                     ,"doc2vec_5"     , "punc_num_..13", "hog_452"
                     ,"hog_649"       , "cnn_10"       ,"cnn_68"
                     ,"doc2vec_6"     , "hog_316"      , "count_vids_mid_high"
                     ,"hog_279"       ,"edge_avg_value", "hog_797"
                     , "Num_Subscribers_Base_low_mid", "doc2vec_17"
                     , "Num_Subscribers_Base_low" ,"doc2vec_1"     , "hog_139"
                     , "doc2vec_19"   , "doc2vec_3"      , "doc2vec_10")
```

6.4 Linear Regression Models

```r
# ============================== #
#  LIBRARIES AND PATH
# ============================== #
set.seed(42)
library(ggplot2)
library(boot)
library(caret)
library(leaps)
library(glmnet)
library(pls)
library(dplyr)

# ============================== #
#  LOADING DATA
# ============================== #
setwd('~/101C/KaggleFinal/')
training <- read.csv("training.csv", header=TRUE)
testing <- read.csv("test.csv", header=TRUE)
cat(dim(training), dim(testing))
youtube_data <- data.frame(training)
attach(youtube_data)

# ================================================ #
#  MULTIPLE LINEAR REGRESSION (model 1)
# ================================================ #
glm.fit <- glm(growth_2_6~., data=youtube_data)
cv.MLR.error <- cv.glm(data.frame(youtube_data,growth_2_6)
                       ,glm.fit, K=10)$delta[1]
summary.MLR <- summary(glm.fit)
trainRMSE.MLR <- sqrt(cv.MLR.error)
trainRMSE.MLR
# RMSE: 1.833286
glm.fit.final <- glm.fit
mlr.pred <- predict(glm.fit.final, testing)

# ================================================ #
#  RIDGE REGRESSION (model 2)
# ================================================ #
x <- model.matrix(growth_2_6~.,youtube_data)
y <- youtube_data$growth_2_6
ridge.fit <- cv.glmnet(x, y, family="gaussian",
                       alpha=0, standardize=TRUE, nfolds=10)
# ridge regression lambda: 0.1278217
ridge.lambda <- ridge.fit$lambda.min
# ridge regression RMSE
trainRMSE.ridge <- sqrt(ridge.fit$cvm[ridge.fit$lambda.min==ridge.fit$lambda])
trainRMSE.ridge
# RMSE: 1.834998
ridge.fit.final <- glmnet(x, y, family="gaussian",
                          alpha=0, standardize=TRUE, lambda = ridge.lambda)
ridge.pred <- predict(ridge.fit.final, newx = model.matrix(~.,testing))

# ================================================ #
#  LASSO REGRESSION (model 3)
# ================================================ #
x <- model.matrix(growth_2_6~.,youtube_data)
y <- youtube_data$growth_2_6
lasso.fit <- cv.glmnet(x, y, family="gaussian",
                       alpha=1, standardize=TRUE, nfolds=10)
# lasso regression lambda: 0.002286282
lasso.lambda <- lasso.fit$lambda.min
# lasso regression RMSE
trainRMSE.lasso <- sqrt(ridge.fit$cvm[lasso.fit$lambda.min==lasso.fit$lambda])
trainRMSE.lasso # RMSE: 1.906478
lasso.fit.final <- glmnet(x, y, family="gaussian",
                          alpha=1, standardize=TRUE, lambda = lasso.lambda)
lasso.pred <- predict(lasso.fit.final, newx = model.matrix(~.,testing))

# ================================================ #
#  Principal Component Regression  (model 4)
# ================================================ #
pcr.fit <- pcr(growth_2_6~., data=youtube_data, scale=TRUE, validation="CV")
# Best number of components: 30
validationplot(pcr.fit, val.type="MSEP")
summary(pcr.fit) # RMSEP: 1.836
pcr.fit.final <- pcr(growth_2_6~., data=youtube_data, scale=TRUE)
xx.pred <- predict(pcr.fit.final, testing, ncomp = 30)


# ================================================ #
#  Partial Least Squares Regression  (model 5)
# ================================================ #
pls.fit <- plsr(growth_2_6~., data=youtube_data,
                scale=TRUE, validation="CV")
# Best number of components: 10
validationplot(pls.fit ,val.type="MSEP")
summary(pls.fit) # RMSE: 1.834
pls.fit.final <- plsr(growth_2_6~., data=youtube_data, scale=TRUE)
pls.pred <- predict(pls.fit.final, newdata = testing,ncomp = 10)

# ================================================ #
#  PRODUCING TEST OUTPUTS (models 1-5)
# ================================================ #
# MLR
```

```r
predictions_mlr <- cbind(testing$id, mlr.pred) %>% as.data.frame()
names(predictions_mlr) <- c("id", "growth_2_6")
write.csv(predictions_mlr, "predictions_mlr_full.csv", row.names = FALSE)
# RIDGE
predictions_ridge <- cbind(testing$id, ridge.pred) %>% as.data.frame()
names(predictions_ridge) <- c("id", "growth_2_6")
write.csv(predictions_ridge, "predictions_ridge_full.csv", row.names = FALSE)
# LASSO
predictions_lasso <- cbind(testing$id, lasso.pred) %>% as.data.frame()
names(predictions_lasso) <- c("id", "growth_2_6")
write.csv(predictions_lasso, "predictions_lasso_full.csv", row.names = FALSE)
# PCR
predictions_pcr <- cbind(testing$id, xx.pred) %>% as.data.frame()
names(predictions_pcr) <- c("id", "growth_2_6")
write.csv(predictions_pcr, "predictions_pcr_full.csv", row.names = FALSE)
# PLS
predictions_pls <- cbind(testing$id, pls.pred) %>% as.data.frame()
names(predictions_pls) <- c("id", "growth_2_6")
write.csv(predictions_pls, "predictions_pls_full.csv", row.names = FALSE)
```

6.5 Random Forest Variable Selection

```r
# ============================== #
#  SETTING UP THE WORKSPACE
# ============================== #
library(gbm) # for boosting
library(glmnet) # for lasso and ridge
library(randomForest) #for randomForest and bagging
library(tree)
library(rattle)
library(caret) # for cv
training <- read.csv("training.csv", header = TRUE)
#for each factor, convert it to numeric
#loop through each name of the variables with the alias "var"
for (var in names(training))
{
  if(is.factor(training[,var]))
  {
    training[,var] = as.numeric(training[,var]) - 1
  }
}

#for each factor, convert it to numeric
#loop through each name of the variables with the alias "var"
for (var in names(test))
{
  if(is.factor(test[,var]))
  {
    test[,var] = as.numeric(test[,var]) - 1
  }
}
test <- read.csv("test.csv", header = TRUE)
rf_model <- randomForest(growth_2_6~., data = training[,-1],
                         mtry = 258/2, ntree=500, importance=TRUE)
rfimportance <- rf_model$importance
```

```r
# ****************************************
# if you have the csv
# ****************************************
rfimportance <- read.csv('~/101C/KaggleFinal/rf_importance.csv', header=TRUE)
best_subset_rf <- rfimportance$X[rfimportance$IncNodePurity>123]
best_subset_rf <- c("PublishedDate"                 ,"Duration"
                      ,"views_2_hours"               ,   "hog_341"
                      ,"hog_641"                     ,   "cnn_10"
                      ,"cnn_12"                      ,   "cnn_17"
                      ,"cnn_19"                      ,   "cnn_25"
                      , "cnn_68"                     ,    "cnn_86"
                      , "cnn_88"                     ,    "cnn_89"
                      , "sd_red"                     ,    "mean_green"
                      , "sd_blue"                    ,    "doc2vec_2"
                      , "doc2vec_3"                  ,    "doc2vec_10"
                      , "doc2vec_13"                 ,    "doc2vec_16"
                      , "punc_num_..28"              ,    "num_words"
                      , "num_chars"                  ,    "num_uppercase_chars"
                      , "num_digit_chars"            ,    "Num_Subscribers_Base_low_mid"
                      , "Num_Subscribers_Base_mid_high", "Num_Views_Base_mid_high"
                      , "avg_growth_low"             ,    "avg_growth_low_mid"
                      , "avg_growth_mid_high"        ,    "count_vids_low_mid"
                      , "count_vids_mid_high")

# Correlation Heatmap for our report:
cor_mtx = round(cor(train[,best_subset_rf[5:14]]), 2)
melted_cor_mtx <- melt(cor_mtx) #reshape it
#draw the heatmap
cor_heatmap = ggplot(data = melted_cor_mtx, aes(x=Var1, y=Var2, fill=value)) + geom_tile()
cor_heatmap = cor_heatmap +
        scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0,
                             limit = c(-1,1), space = "Lab", name="Pearson\nCorrelation") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, vjust = 1, size = 12, hjust = 1))
cor_heatmap
```

6.6 Data Processing for Date and Time Variables

```r
# ============================== #
#  LIBRARIES AND PATH
# ============================== #
library(dplyr)
library(gbm)
library(glmnet)
library(randomForest)
library(caret)
library(tidyr)
library(lubridate)

# ============================== #
#  PROCESSING TRAINING DATA
# ============================== #
training <- training %>% separate(PublishedDate, c("Date", "Time"), sep = " ")
# convert date and time columns to posix type
training$Date <- as.Date(training$Date, "%m/%d/%Y")
training$Time <- format(strptime(training$Time, "%H:%M"),"%H:%M")
```

```
# now we want to create a new column for how many minutes it has been since midnight
Hour <- training$Time %>% strptime(format = "%H:%M") %>% hour()
Minute <- training$Time %>% strptime(format = "%H:%M") %>% minute()
training$Time <- Hour*60 + Minute # creates new column
head(training[,c(1:4,261)]) # sanity check
# for both training and testing, the earliest date is April 11th, 2020 (start date)
# create a new variable that gives us the number of days since 2020-04-11
training$daysSince0411 <- difftime(training$Date, as.Date("2020-04-11"),
                                   units = "days") %>% as.numeric
head(daysSince0411) # sanity check
training$dayOfWeek <- training$daysSince0411 %% 7
train <- training[,-c(1,2)] # this gets rid of the id and unused date column
ead(train) #sanity check
write.csv(train, "train.csv", row.names = FALSE) #this writes the new csv

# ============================== #
#  PROCESSING TESTING DATA
# ============================== #
test <- test %>% separate(PublishedDate, c("Date", "Time"), sep = " ")
#convert date and time columns to posix type
test$Date <- as.Date(test$Date, "%m/%d/%Y")
test$Time <- format(strptime(test$Time, "%H:%M"),"%H:%M")
#now we want to create a new column for how many minutes it has been since midnight
Hour <- test$Time %>% strptime(format = "%H:%M") %>% hour()
Minute <- test$Time %>% strptime(format = "%H:%M") %>% minute()
test$Time <- Hour*60 + Minute #creates new column
head(test[,c(1:4,260)]) #sanity check
# for both datasets, the earliest date is April 11th, 2020. This will be our start date
# create a new variable that gives us the number of days since 2020-04-11
test$daysSince0411 <- difftime(test$Date, as.Date("2020-04-11"), units = "days") %>% as.numeric
head(daysSince0411) #sanity check
test$dayOfWeek <- test$daysSince0411 %% 7
testing <- test[,-2] #this keeps id column for test set and deletes unused date column
head(testing) #sanity check
write.csv(testing, "testing.csv", row.names = FALSE) #this writes the new csv
```

6.7 Random Forest Hyperparameter Grid Search

```
# ============================== #
#  LIBRARIES AND PATH
# ============================== #
library(randomForest)
setwd("~/101C/KaggleFinal/")
training <- read.csv("CSVs/trainclean.csv", header = TRUE)
testing <- read.csv("CSVs/testclean.csv", header = TRUE)
rfimportance <- read.csv('CSVs/rfimportance.csv', header = TRUE)
rfimportance$X <- as.character(rfimportance$X)

# ============================== #
#  SELECTING BEST SUBSET
# ============================== #
best_subset_rf <- rfimportance$X[rfimportance$X.IncMSE > .01] #can change the value
best_subset_rf <- c(best_subset_rf,"daysSince0411","Time","growth_2_6")
# Def a function
'%notin%' <- Negate('%in%')
best_subset_rf <- best_subset_rf[which(best_subset_rf %notin%
```

```r
        c("PublishedDate", "mean_green", "mean_red",
          "num_words", "cnn_19", "cnn_68", "cnn_88"))]
training["daysSince0411"] <- training["daysSince0411"] %% 7
nodesizes <- seq(13,16,1) #adjust this
maxnodes <- seq(430,466,5) #adjust this
numtrees <- c(1024) #adjust this

# ============================== #
#  GRID SEARCH ON PARAMETERS
# ============================== #
#Errors
rf_errs <- c()
cv_errs <- c()
for(k in 1:length(numtrees)){
    for(i in 1:length(nodesizes)){
        for(j in 1:length(maxnodes)){
            rf.train <- randomForest(growth_2_6~., data = training[, best_subset_rf],
                                     mtry = length(best_subset_rf)-1,
                                     ntree=numtrees[k], nodesize= nodesizes[i],
                                     maxnodes = maxnodes[j], importance=FALSE)
            #calculate training RMSE
            yhat <- predict(rf.train, newdata = training)
            rf_errs <- c(rf_errs, sqrt(mean((yhat - training$growth_2_6)^2)))
            # get cv_error and report to kaggle
            cv_errs <- c(cv_errs, sqrt(rf.train$mse[numtrees[k]]))
            print(paste("DONE:",numtrees[k], nodesizes[i],maxnodes[j],
                        sqrt(rf.train$mse[numtrees[k]])))
        }
    }
}
#obtain test errors
errors <- cbind(rep(nodesizes,length(maxnodes)),
                rep(maxnodes, each = length(nodesizes)), rf_errs, cv_errs)
errors
rf_errs

# ============================== #
#  TESTING ON DATA
# ============================== #
rf.train <- randomForest(growth_2_6~., data = training[, best_subset_rf],
               mtry = length(best_subset_rf)-1, ntree=1024, nodesize=25,
               maxnodes = 420, importance=FALSE)
pls.pred <- predict(rf.train, newdata = testing)
rf.train$mse[512]
library(dplyr)
predictions_pls <- cbind(testing$id, pls.pred) %>% as.data.frame()
names(predictions_pls) <- c("id", "growth_2_6")
write.csv(predictions_pls, "CSVs/predictions_rfclean.csv", row.names = FALSE)
```

6.8 Random Forest Models

```r
# ============================== #
#  LIBRARIES AND PATH
# ============================== #
library(dplyr)
library(randomForest) #for randomForest and bagging
library(caret) # for cv

# ============================== #
#  PULLING BEST SUBSET
#  FROM RF VARIABLE SELECTION
#  SCRIPT AND THRESHOLDS
# ============================== #
#After PublishedDate update

train <- read.csv("train.csv", header = TRUE)
setwd("/Documents and Settings/turke/Documents/101C/KaggleFinal/")
rfimportance <- read.csv('CSVs/rfimportance.csv', header = TRUE)
rfimportance$X <- as.character(rfimportance$X)
best_subset_rf <- rfimportance$X[rfimportance$X.IncMSE > 0.01] #can change the value
best_subset_rf <- c(best_subset_rf,"daysSince0411","Time","growth_2_6")

# ============================== #
#  BEST RANDOM FOREST SUBSET
#  REDUCED COLLINEAR VARIABLES
# ============================== #
# Def a function
'%notin%' <- Negate('%in%')
best_subset_rf <- best_subset_rf[which(best_subset_rf %notin%
      c("PublishedDate", "mean_green", "mean_red","num_words",
        "cnn_19", "cnn_68", "cnn_88", "sd_pixel_val","sd_red","sd_green"))]
cor_mtx = round(cor(train[best_subset_rf]), 2)
cor_mtx

# ============================== #
# BEST BAGGING MODEL
# ============================== #
#Bagging Model
rf_model <- randomForest(formula = growth_2_6 ~ .,
    data = train[, best_subset_rf], mtry = floor(length(best_subset_rf) - 1),
    ntree = 500, sample = TRUE, importance = TRUE)
rf_model$importance
print(rf_model)
varImpPlot(rf_model)
tree_min <- which.min(rf_model$mse)
tree_min #500
sqrt(rf_model$mse[tree_min]) #how does this compare to our Kaggle public scores?
#calculate training RMSE... this is not very important anymore
yhat <- predict(rf_model, newdata = train)
rf_errs <- mean((yhat - train$growth_2_6)^2)
sqrt(rf_errs)
# get cv_error and report to kaggle
sqrt(tail(mean(rf_model$mse)))
sqrt(rf_model$mse[1000]) # to get the last OOB RMSE value (change value if not ntree=500)
#to create predictions
test.predicts = predict(rf_model, newdata = test[,-1])
predictions = cbind(test$id, test.predicts) %>% as.data.frame()
names(predictions) <- c("id", "growth_2_6")
#creating csv file to turn in
write.csv(predictions, "predictions51.csv", row.names = FALSE)
# change csv name per prediction

# ============================== #
#  BEST RANDOM FOREST SUBSET
#  REDUCED COLLINEAR VARIABLES
# =============================== #
best_subset_rf <- rfimportance$X[rfimportance$X.IncMSE > .008] #can change the value
best_subset_rf <- c(best_subset_rf,"daysSince0411","Time","growth_2_6")
# Def a function
'%notin%' <- Negate('%in%')
#to remove collinearity
best_subset_rf <- best_subset_rf[which(best_subset_rf %notin% c("PublishedDate",
    "mean_green", "mean_red","num_words", "cnn_19",
    "cnn_68", "cnn_88", "sd_pixel_val","sd_red","sd_green"))]

# =============================== #
#  RANDOM FOREST MODELS
# =============================== #
rf_model <- randomForest(formula = growth_2_6 ~ ., data = train[, best_subset_rf],
              mtry = floor(length(best_subset_rf) - 1)/2, ntree = 1000, maxnode = 450,
              nodesize = 15, sample = TRUE, importance = TRUE)
rf_model$importance # this shows variable importance
#and how much mse would increase by if it is removed. Make sure all are positive values!

#get some more info on the rf
print(rf_model)
varImpPlot(rf_model)
tree_min <- which.min(rf_model$mse)
tree_min #500
sqrt(rf_model$mse[tree_min]) #how does this compare to our Kaggle public scores?
#calculate training RMSE... this is not very important anymore
yhat <- predict(rf_model, newdata = train)
rf_errs <- mean((yhat - train$growth_2_6)^2)
sqrt(rf_errs)
# get cv_error and report to kaggle
sqrt(tail(mean(rf_model$mse)))
sqrt(rf_model$mse[1000]) # to get the last OOB RMSE value (change value if not ntree=500)
#to create predictions
test.predicts = predict(rf_model, newdata = test[,-1])
predictions = cbind(test$id, test.predicts) %>% as.data.frame()
names(predictions) <- c("id", "growth_2_6")
#creating csv file to turn in
write.csv(predictions, "predictions75.csv", row.names = FALSE) #change csv name
```

7 Statement of Contributions

Richard worked mainly on the linear regression model results. He tried all variations of linear regression models on the dataset and made multiple submissions. He also worked on some random forest hyperparameter search material and leverage analysis. He is also responsible for knitting and compiling the final report in an RMD knitted file. He also contributed to the completion of the presentation.

Cassandra wrote the exploratory data analysis code as well as templates for boosting, bagging, random forest models, and hyperparameter tuning for random forest. She also wrote the code to convert the `publishedDate` predictor into two numerical variables that were important for the team's models. She also contributed most of the Kaggle submissions and wrote most of the content for the report and presentation.

Yeryeong explored the boosting and the ridge regression model for the final report. She made submissions on Kaggle for these models. She was also responsible for producing the slides and helping with creating the video for the presentation.