Predicting Early Growth Patterns on YouTube

Group Team
Allan Chang (allanchangst@outlook.com) Yutaro Kobayashi (yutarokobayashi99@gmail.com)
Ajay Kallepalli (ajay.r.kallepalli@gmail.com)

I. Introduction

The goal of our model was to use certain distinguishing features of YouTube videos to best predict the percentage change in views on a video between the second and sixth hour since its publishing, labeled as *growth_2_6* in the dataset used. The early stages of a video being published is especially important to determine how well the video will do as well as how well the channel posting it is doing. Provided by Ritvik Kharkar, the dataset included 7242 observations and 259 predictors regarding video features. These 259 predictors included thumbnail image features, video title features, channel features, and other features such as published date.

II. Methodology

a) Preprocessing

Our initial step was to remove any missing data or NA values, to which we found there were none. We then looked into each feature and their descriptions in an attempt to understand and predict which features will contribute the most to views. Initially, we noticed that the time the video is published may provide insight into viewership because there are certain time, day, and month intervals where people tend to watch more YouTube (Gielen, 2015). Therefore, we split the *PublishedDate* feature into individual columns of *day* (day of the month), *weekday* (0 - 6 numeric values indicating the day of the week), *weekend* (binary variable where 1 is weekend and 0 is weekday), *minute* (minute of the hour), *min_2hours* (time of the day in minutes plus two hours to represent the start time of our response), and *min_day* (time of the day in minutes). We also observed that we can reformat *Num_Subscribers*, *Num_Views*, *avg_growth*, and *count_vids* each into ordered factor with levels from 1 to 4, where 1 is the lowest and 4 is the highest, thus consolidating the information into one column each.

Subsequently, we removed any columns that had the same value for all observations because these predictors would provide no extra information, such as *cnn_0*. We also removed any highly unbalanced columns (i.e. extreme majority of the observations is one certain value) for the same reason. We also used Cook's Distance as a method to remove any influential outliers to prevent the unusual points from affecting the overall model. This reduced the number of observations from 7242 to 6828. We ran an initial test random forest before and after removing the outliers and found that the RMSE improved after removing the outliers. Therefore, we decided that there are sufficient observations remaining for model training.

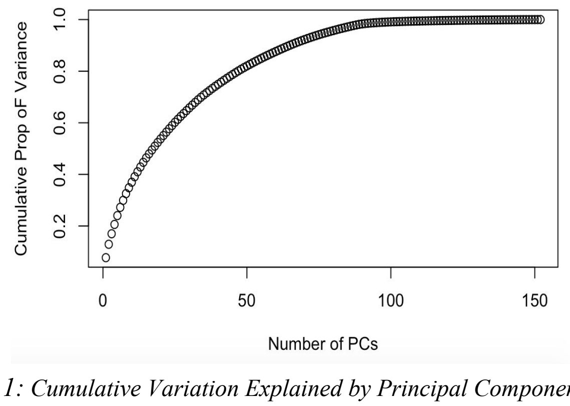


*Figure 1: Cumulative Variation Explained by Principal Components of hog_*

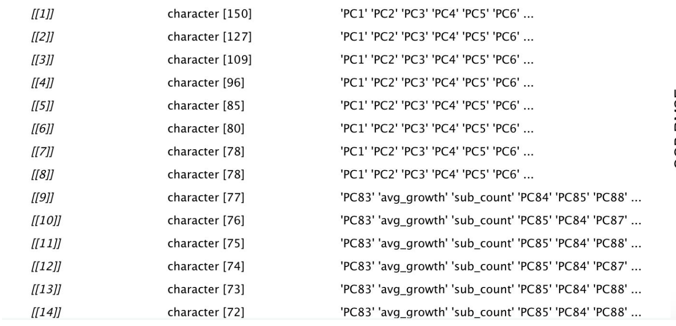
We then ran PCA on subsets of the features. We chose subsets that were correlated to each other, were of a similar type of feature, and features that were already hard to interpret. PCA was run on 152 hog features where 76 components explaining 95% of the variation were kept. This can be seen in Figure 1 above. PCA was run separately on 12 cnn features where 7 components explaining 98% of the variation were kept. Finally we ran another separate PCA on 9 of the color related features, keeping 4 components that explained 94% of the variation. These principle components were substituted for each subset of features that PCA was applied to.

Finally, we removed/combined any remaining variables with correlation above 0.9 to avoid multicollinearity. For example, *num_chars* and *num_words* have a correlation of 0.91. We divided *num_chars* by *num_words* to create the mean number of characters per word and removed *num_chars* and *num_words*.

b) Statistical Model

150 total features remained after preprocessing. We ran several different methods on these 150 models including lasso regression, ridge regression, and partial least squares. We also ran knn regression after using the lasso and ridge to shrink coefficients or after further reducing dimensions using partial least squares. However these methods resulted in training RMSE values above 1.8 using cross validation(Refer Appendix 3.1 for methods used other than final model). With the knowledge that Random forests are able to efficiently handle large sets of predictors without a large risk of overfitting, we ran Random forest on the remaining 150 features. This resulted in a training RMSE of around 1.44 using the out of bag method. Given how well random forest performed with all 150 features in comparison to previous models, we decided to use this model as a baseline model to be further improved upon.

We then decided to recursively run a random forest model on the 150 features, each time removing any features that had a negative % MSE Increase value. Each model was evaluated using the out of bag method and tuned for the optimal number of variables to try at each split. This resulted in a random forest model with 78 remaining features of which all 78 had a positive % MSE Increase value. We then continued to recursively run a random forest. However this time, the predictor with the lowest % MSE Increase value was removed after each iteration. The reduction of dimensionality through PCA proved vital for computational efficiency during this part of the project. A set of predictors was then chosen based on the training RMSE. The Figure 2 below shows the first 150 features and the remaining features after each iteration. Figure 3 then shows the corresponding training OOB RMSE evaluated using the out of bag method. Although Random Forests are capable of handling a large number of predictors, we made an effort to choose smaller sets of predictors with low training RMSE to reduce complexity, and thus reduce the variance of our model.



```
[[1]]     character [150]   'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[2]]     character [127]   'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[3]]     character [109]   'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[4]]     character [96]    'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[5]]     character [85]    'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[6]]     character [80]    'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[7]]     character [78]    'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[8]]     character [78]    'PC1' 'PC2' 'PC3' 'PC4' 'PC5' 'PC6' ...
[[9]]     character [77]    'PC83' 'avg_growth' 'sub_count' 'PC84' 'PC85' 'PC88' ...
[[10]]    character [76]    'PC83' 'avg_growth' 'sub_count' 'PC85' 'PC84' 'PC87' ...
[[11]]    character [75]    'PC83' 'avg_growth' 'sub_count' 'PC85' 'PC84' 'PC88' ...
[[12]]    character [74]    'PC83' 'avg_growth' 'sub_count' 'PC85' 'PC84' 'PC87' ...
[[13]]    character [73]    'PC83' 'avg_growth' 'sub_count' 'PC85' 'PC84' 'PC88' ...
[[14]]    character [72]    'PC83' 'avg_growth' 'sub_count' 'PC85' 'PC84' 'PC88' ...
```

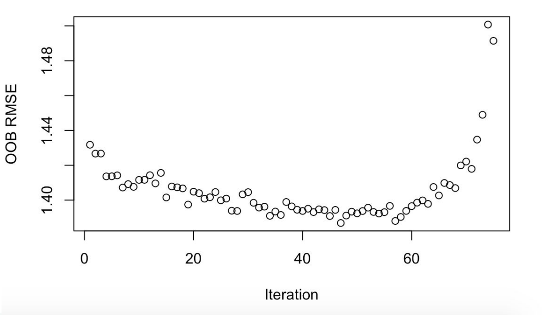


Figure 2: Features Used for Iterations

Figure 3: Training OOB RMSE for Each Iteration

III. Results

Our final model used a total of 27 predictors, as seen in Figure 4, and achieved a training RMSE in R of 1.393805 and a test RMSE of 1.39134 and was ranked 9th on the public leaderboard. Our second chosen test RMSE was 1.39464. The strongest driver of predicted growth was average growth of other videos from the channel, which is expected because it is highly correlated with our response. The other most important features include PC83, PC84, and PC85, which correspond to the *cnn* (Convolutional Neural Network) features in the original dataset.

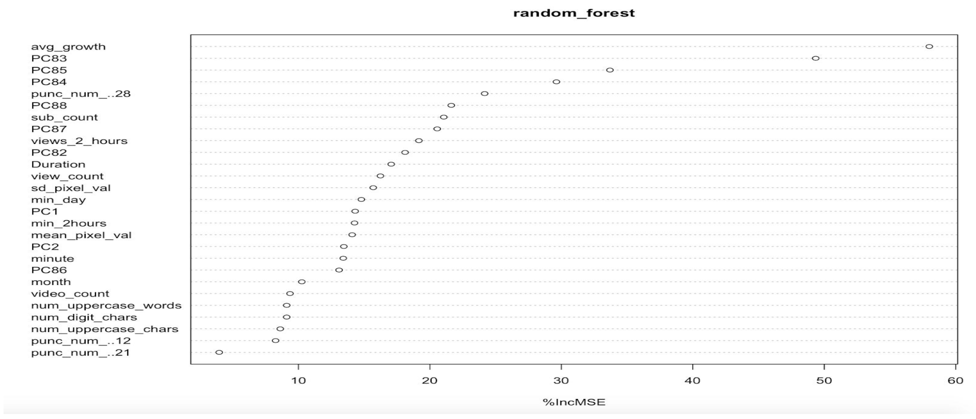


Figure 4: Importance Plot of the Final Model

IV. Conclusions

The final random forest model we used performed extremely well in both the public and private leaderboard with an RMSE of 1.39134 and 1.39224, respectively. This difference is minimal and can be explained due to variance which shows that the model was consistent and the performance was not a result of overfitting. Our model performed better when compared to our initial lasso regression, ridge regression, KNN, and partial least squares models.

We believe that random forest performed the best because it was best suited to handle our post-processed data. Because random forest only considers a random sample of a small number of variables at each split instead of all possible variables, it is able to run on larger datasets faster which was necessary as our post-processed data set was extremely large and included 150 variables. This enabled us to perform multiple iterations of the model in order to find the best subset of variables. We also know that random forest works well with non-linear data and has a lower risk of overfitting as it takes the average of all decision trees. Moreover, by only choosing a random subset of predictors at each split, random forest is able to add diversity to the trees. This ultimately leads to trees being less correlated, which can also lead to lower variability in our model. This is shown in the minor differences between our public and private leaderboard scores. These benefits were factors in random forest outperforming the other models. In spite of its impressive performance, we think the model could have been slightly improved by further interpreting the importance plot. For instance, PCA 82-88, several of which are considered highly important by our final model, are components created from the *cnn* features. As PCA leads to loss in information, running the models with the original cnn features could have led to an even better performance. We also believe that a finer tuning of random forest's arguments, such as number of nodes and number of grown trees, would have led to better results.

V. Appendix

See Below

VI. Statement of Contribution

Sheng-Ting (Allan) Chang: Preprocessed data and attempted to reduce the number of features through lasso and random forest variable importance plots, and ultimately tried random forest as the best model for prediction.

Yutaro Kobayashi: Preprocessed data using PCA and attempted several models including lasso regression, ridge regression, partial least squares, knn regression, and a combination of the above. Built upon the idea of using Random forest by deciding to recursively remove features based on importance.

Ajay Kallepalli: Performed an initial probe of various models excluding Random Forest such as SVM and Linear Regression to gauge viability and eliminate candidate models based on results.

Finally, we combined our work and finished the report and presentation together.

VII. References

Gielen, M. (2015, January 12). *Want To Know The Best Days And Times To Post YouTube Videos? Here's A Yearly Calendar.* Tubefilter. http://www.tubefilter.com/2015/01/12/best-days-times-to-post-youtube-videos-yearly-calendar/.

101C_Final_Appendix

Appendix

Libraries

```
library(stringr)
library(reshape2)
library(ggplot2)
library(caret)
library(glmnet)
library(pls)
library(randomForest)
library(knitr)
```

Methodology : Preprocessing

1.1 Simple Feature Creation

```
setwd("~/Desktop")
prj <- read.csv("101C_Final-Project.csv", row.names = 1, na.strings=c("","NA"))

### Published Date
date_chr <- as.character(prj$PublishedDate)
# Month
month <- str_extract(date_chr, "^[\\d]{1,2}")
month <- factor(month, labels = c("April", "May", "June", "July", "August", "Septemb
er"))
month <- as.numeric(month)
# Day
day <- str_extract(date_chr, "(?<=/)\\d{1,2}(?=/)")
day <- as.numeric(day)
# Weekday
date <- as.Date(str_extract(date_chr, ".*(?=\\s)"), "%m/%d/%y")
weekday <- as.POSIXlt(date)$wday
# Weekend or Not
weekend <- rep(0, length(date_chr))
weekend[weekday == 0 | weekday == 6] <- 1 # 1 if weekend
# Time
time <- str_extract(date_chr, "\\d{1,2}:\\d{1,2}")
# Hour
hour <- str_extract(date_chr, "\\d{1,2}(?=:)")
hour <- as.numeric(hour)
# Minute
minute <- str_extract(date_chr, "(?<=:)\\d{1,2}")
minute <- as.numeric(minute)
# xth minute of day
min_day <- (hour * 60) + minute
# xth minute after 2 hours
min_2hours <- min_day + 120
for (i in 1:length(min_2hours)) {
  if(min_2hours[i] >= 1440) min_2hours[i] <- min_2hours[i] - 1440
}
```

```
### Combine low/high binary features and removing them
# subscribers
a <- b <- rep(0,nrow(prj))
a[as.logical(prj$Num_Subscribers_Base_low_mid)] <- 2
b[as.logical(prj$Num_Subscribers_Base_mid_high)] <- 3
sub_count <- prj$Num_Subscribers_Base_low + a + b
sub_count[sub_count==0] <- 4
# views
a <- b <- rep(0,nrow(prj))
a[as.logical(prj$Num_Views_Base_low_mid)] <- 2
b[as.logical(prj$Num_Views_Base_mid_high)] <- 3
view_count <- prj$Num_Views_Base_low + a + b
view_count[view_count==0] <- 4
# average growth
a <- b <- rep(0,nrow(prj))
a[as.logical(prj$avg_growth_low_mid)] <- 2
b[as.logical(prj$avg_growth_mid_high)] <- 3
avg_growth <- prj$avg_growth_low + a + b
avg_growth[avg_growth==0] <- 4
# video count
a <- b <- rep(0,nrow(prj))
a[as.logical(prj$count_vids_low_mid)] <- 2
b[as.logical(prj$count_vids_mid_high)] <- 3
video_count <- prj$count_vids_low + a + b
video_count[video_count==0] <- 4
# removal
prj <- prj[,-(247:258)]

### Combine New Features / Remove Published Date
prj <- cbind(sub_count,view_count,avg_growth,video_count,month,
             day,weekday,weekend,minute,min_2hours,min_day,prj)
prj <- prj[,-which(colnames(prj) == "PublishedDate")]
```

1.2 Data Cleaning

```
### Remove featuers that have same values for all observations
keep <- sapply(prj, function(x){length(unique(x))>1})
prj <- prj[,keep]
### Remove features that are very unbalanced
remove <- sapply(prj, function(x){length(table(x)) == 2 & any(table(x)<10)})
prj <- prj[,!remove]

### Finding outliers and removing them
prj <- prj[-c(6514),]
lm_fit <- lm(growth_2_6~., prj)
cooks <- cooks.distance(lm_fit)
out <- cooks > (4/nrow(prj))
prj <- prj[!out,]
```

1.3 Same Preprocessing on test data

```
setwd("~/Desktop")
tst <- read.csv("101C_Final_Test.csv", row.names = 1, na.strings=c("","NA"))

### Data Exploration ``
```

```r
### Published Date
date_chr <- as.character(tst$PublishedDate)
# Month
month <- str_extract(date_chr, "^[\\d]{1,2}")
month <- factor(month, labels = c("April", "May", "June", "July", "August", "Septemb
er"))
month <- as.numeric(month)
# Day
day <- str_extract(date_chr, "(?<=/)\\d{1,2}(?=/)")
day <- as.numeric(day)
# Weekday
date <- as.Date(str_extract(date_chr, ".*(?=\\s)"), "%m/%d/%y")
weekday <- as.POSIXlt(date)$wday
# Weekend or Not
weekend <- rep(0, length(date_chr))
weekend[weekday == 0 | weekday == 6] <- 1 # 1 if weekend
# Time
time <- str_extract(date_chr, "\\d{1,2}:\\d{1,2}")
# Hour
hour <- str_extract(date_chr, "\\d{1,2}(?=:)")
hour <- as.numeric(hour)
# Minute
minute <- str_extract(date_chr, "(?<=:)\\d{1,2}")
minute <- as.numeric(minute)
# xth minute of day
min_day <- (hour * 60) + minute
# xth minute after 2 hours
min_2hours <- min_day + 120
for (i in 1:length(min_2hours)) {
  if(min_2hours[i] >= 1440) min_2hours[i] <- min_2hours[i] - 1440
}
# What part of the hour
# Part of day

### Combine low/high binary features and removing them
# subscribers
a <- b <- rep(0,nrow(tst))
a[as.logical(tst$Num_Subscribers_Base_low_mid)] <- 2
b[as.logical(tst$Num_Subscribers_Base_mid_high)] <- 3
sub_count <- tst$Num_Subscribers_Base_low + a + b
sub_count[sub_count==0] <- 4
# views
a <- b <- rep(0,nrow(tst))
a[as.logical(tst$Num_Views_Base_low_mid)] <- 2
b[as.logical(tst$Num_Views_Base_mid_high)] <- 3
view_count <- tst$Num_Views_Base_low + a + b
view_count[view_count==0] <- 4
# average growth
a <- b <- rep(0,nrow(tst))
a[as.logical(tst$avg_growth_low_mid)] <- 2
b[as.logical(tst$avg_growth_mid_high)] <- 3
avg_growth <- tst$avg_growth_low + a + b
avg_growth[avg_growth==0] <- 4
# video count
a <- b <- rep(0,nrow(tst))
a[as.logical(tst$count_vids_low_mid)] <- 2
b[as.logical(tst$count_vids_mid_high)] <- 3
```

```
video_count <- tst$count_vids_low + a + b
video_count[video_count==0] <- 4

### Combine New Features / Remove Published Date
tst <- cbind(sub_count,view_count,avg_growth,video_count,month,
             day,weekday,weekend,minute,min_day,min_2hours,tst)
tst <- tst[,-which(colnames(tst) == "PublishedDate")]

### keeping features in training data
tst <- tst[,colnames(prj)[-ncol(prj)]]
```

1.4 Running PCA on subsets of the data

```
PCA <- rbind(prj[,-ncol(prj)],tst)

# cnn pca
pca_cnn <- princomp(PCA[,which(colnames(PCA) == "cnn_9"):which(colnames(PCA) == "cnn
_89")], cor = TRUE)
summary(pca_cnn)
var <- pca_cnn[["sdev"]]^2
var_prop <- var/sum(var)
plot(cumsum(var_prop))
cnn <- pca_cnn[["scores"]][,1:7]

PCA <- PCA[,-(which(colnames(PCA) == "cnn_9"):which(colnames(PCA) == "cnn_89"))]
PCA <- cbind(cnn, PCA)

### hog pca
pca_hog <- princomp(PCA[,which(colnames(PCA) == "hog_0"):
                          which(colnames(PCA) == "hog_863")], cor = TRUE)
summary(pca_hog)
var <- pca_hog[["sdev"]]^2
var_prop <- var/sum(var)
plot(cumsum(var_prop), xlab = "Number of PCs", ylab = "Cumulative Prop oF Variance")
hog <- pca_hog[["scores"]][,1:76]

PCA <- PCA[,-(which(colnames(PCA) == "hog_0"):
                which(colnames(PCA) == "hog_863"))]
PCA <- cbind(hog, PCA)

### pixel color pca
pca_pixel <- princomp(PCA[,which(colnames(PCA) == "mean_red"):
                            which(colnames(PCA) == "sd_blue")], cor = TRUE)
summary(pca_pixel)
var <- pca_pixel[["sdev"]]^2
var_prop <- var/sum(var)
plot(cumsum(var_prop))
pixel <- pca_pixel[["scores"]][,1:4]

PCA <- PCA[,-(which(colnames(PCA) == "mean_red"):
                which(colnames(PCA) == "sd_blue"))]
PCA <- cbind(pixel, PCA)

# renaming principle components
colnames(PCA)[1:88] <- paste("PC", as.character(1:88),sep = "")
```

1.5 Final Data Cleaning

```r
# removing/combining features with correlation > 0.9
corr <- abs(cor(PCA[,100:151]))
above <- which(corr >= 0.9, arr.ind = TRUE)
above <- above[!(above[,1] == above[,2]),]
int <- matrix(0,nrow(PCA),0)
for (i in seq(from = 1, by = 2, length.out = nrow(above)/2)) {
  int <- cbind(int,PCA[,rownames(above)[i]]*PCA[,rownames(above)[i+1]])
}
### creating mean characters per word from two highly corrleated features
int[,3] <- PCA$num_chars/PCA$num_words
colnames(int) <- paste("INT", as.character(1:ncol(int)),sep = "")
PCA <- PCA[,-which(colnames(PCA) %in% rownames(above))]
PCA <- cbind(PCA, int)

# making appropriate features into ordered factors and factors
PCA$avg_growth <- factor(PCA$avg_growth, ordered = TRUE)
PCA$sub_count <- factor(PCA$sub_count, ordered = TRUE)
PCA$view_count <- factor(PCA$view_count, ordered = TRUE)
PCA$video_count <- factor(PCA$video_count, ordered = TRUE)
PCA$month <- factor(PCA$month)
PCA$weekday <- factor(PCA$weekday)

# making an original copy
PCA_copy <- PCA
```

```r
# splitting back into training and test data
train_PCA <- PCA[1:nrow(prj),]
train_PCA <- cbind(train_PCA, "growth_2_6" = prj$growth_2_6)
test_PCA <- PCA[(nrow(prj)+1):nrow(PCA),]
```

Methodology : Statistical Model

2.1 Removing features with negative % MSE value by while loop

```r
# setting out of bag method to evaluate
oob_train_control <- trainControl(method="oob", savePredictions = TRUE)
last_rmse <- matrix(0,0,3)
last_fac <- list()
colnames(last_rmse) <- c("caret","randomforest","average")
value <- TRUE
i <- 1


while(value){
  train_PCA <- PCA[1:nrow(prj),]
  train_PCA <- cbind(train_PCA, "growth_2_6" = prj$growth_2_6)
  rf <- train(growth_2_6 ~ . , data = train_PCA, method = "rf" ,
              ntree = 200, importance = TRUE, trControl = oob_train_control)
  rf_tree_imp <- varImp(rf, scale = FALSE)
  rf_tree_imp <- rf_tree_imp[["importance"]]
  month_ind <- which(grepl("month",rownames(rf_tree_imp)))
  if(sum(month_ind) != 0){
    month_sum <- sum(rf_tree_imp[month_ind,])
    rf_tree_imp <- rf_tree_imp[-month_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "month" = month_sum)
  }
  weekday_ind <- which(grepl("weekday",rownames(rf_tree_imp)))
  if(sum(weekday_ind) != 0){
    weekday_sum <- sum(rf_tree_imp[weekday_ind,])
    rf_tree_imp <- rf_tree_imp[-weekday_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "weekday" = weekday_sum)
  }
  sub_ind <- which(grepl("sub_count",rownames(rf_tree_imp)))
  if(sum(sub_ind) != 0){
    sub_sum <- sum(rf_tree_imp[sub_ind,])
    rf_tree_imp <- rf_tree_imp[-sub_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "sub_count" = sub_sum)
  }
  view_ind <- which(grepl("view_count",rownames(rf_tree_imp)))
  if(sum(view_ind) != 0){
    view_sum <- sum(rf_tree_imp[view_ind,])
    rf_tree_imp <- rf_tree_imp[-view_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "view_count" = view_sum)
  }
  vid_ind <- which(grepl("video_count",rownames(rf_tree_imp)))
  if(sum(vid_ind) != 0){
    vid_sum <- sum(rf_tree_imp[vid_ind,])
    rf_tree_imp <- rf_tree_imp[-vid_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "video_count" = vid_sum)
  }
  avg_ind <- which(grepl("avg_growth",rownames(rf_tree_imp)))
  if(sum(avg_ind) != 0){
    avg_sum <- sum(rf_tree_imp[avg_ind,])
    rf_tree_imp <- rf_tree_imp[-avg_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "avg_growth" = avg_sum)
  }
  caret_rmse <- rf[["results"]][["RMSE"]][2]
  m <- rf$bestTune[1,1]
  random_forest <- randomForest(growth_2_6 ~ . , data = train_PCA, mtry = m, ntree =
 200)
  rf_rmse <- mean((random_forest$predicted - random_forest$y)^2)
  last_rmse <- rbind(last_rmse, c(caret_rmse,rf_rmse, (caret_rmse + rf_rmse)/2))
  last_fac[[i]] <- colnames(PCA)
  k <- rownames(rf_tree_imp)[which(rf_tree_imp[,1] >= 0)]
  PCA <- PCA[,k]
  i <- i+1
  value <- any(rf_tree_imp[,1] < 0)
}
```

2.2 Removing feature low negative % MSE value by while loop until 10 featurse remaining

```
last_rmse_2 <- matrix(0,0,3)
last_fac_2 <- list()
i <- 1

while(ncol(PCA) != 10){
  train_PCA <- PCA[1:nrow(prj),]
  train_PCA <- cbind(train_PCA, "growth_2_6" = prj$growth_2_6)
  rf <- train(growth_2_6 ~ . , data = train_PCA, method = "rf" ,
              ntree = 200, importance = TRUE, trControl = oob_train_control)
  rf_tree_imp <- varImp(rf, scale = FALSE)
  rf_tree_imp <- rf_tree_imp[["importance"]]
  month_ind <- which(grepl("month",rownames(rf_tree_imp)))
```

```r
  if(sum(month_ind) != 0){
    month_sum <- sum(rf_tree_imp[month_ind,])
    rf_tree_imp <- rf_tree_imp[-month_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "month" = month_sum)
  }
  weekday_ind <- which(grepl("weekday",rownames(rf_tree_imp)))
  if(sum(weekday_ind) != 0){
    weekday_sum <- sum(rf_tree_imp[weekday_ind,])
    rf_tree_imp <- rf_tree_imp[-weekday_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "weekday" = weekday_sum)
  }
  sub_ind <- which(grepl("sub_count",rownames(rf_tree_imp)))
  if(sum(sub_ind) != 0){
    sub_sum <- sum(rf_tree_imp[sub_ind,])
    rf_tree_imp <- rf_tree_imp[-sub_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "sub_count" = sub_sum)
  }
  view_ind <- which(grepl("view_count",rownames(rf_tree_imp)))
  if(sum(view_ind) != 0){
    view_sum <- sum(rf_tree_imp[view_ind,])
    rf_tree_imp <- rf_tree_imp[-view_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "view_count" = view_sum)
  }
  vid_ind <- which(grepl("video_count",rownames(rf_tree_imp)))
  if(sum(vid_ind) != 0){
    vid_sum <- sum(rf_tree_imp[vid_ind,])
    rf_tree_imp <- rf_tree_imp[-vid_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "video_count" = vid_sum)
  }
  avg_ind <- which(grepl("avg_growth",rownames(rf_tree_imp)))
  if(sum(avg_ind) != 0){
    avg_sum <- sum(rf_tree_imp[avg_ind,])
    rf_tree_imp <- rf_tree_imp[-avg_ind,1, drop = FALSE]
    rf_tree_imp <- rbind(rf_tree_imp, "avg_growth" = avg_sum)
  }
  caret_rmse <- rf[["results"]][["RMSE"]][2]
  m <- rf$bestTune[1,1]
  random_forest <- randomForest(growth_2_6 ~ . , data = train_PCA, mtry = m, ntree =
 200)
  rf_rmse <- mean((random_forest$predicted - random_forest$y)^2)
  last_rmse_2 <- rbind(last_rmse_2, c(caret_rmse,rf_rmse, (caret_rmse + rf_rmse)/2))
  ordered_imp <- rf_tree_imp[order(rf_tree_imp[,1], decreasing = TRUE),1, drop = FAL
SE]
  ordered_imp <- ordered_imp[-nrow(ordered_imp),1, drop = FALSE]
  d <- rownames(ordered_imp)
  last_fac_2[[i]] <- colnames(PCA)
  i <- i + 1
  PCA <- PCA[,d]
}
```

2.3 Choosing final 27 predictors based on stored rmse values above and tuned m value

```r
PCA <- PCA_copy[,last_fac_2[[52]]]
train_PCA <- PCA[1:nrow(prj),]
train_PCA <- cbind(train_PCA, "growth_2_6" = prj$growth_2_6)

recommended.mtry <- 10
```

```
tunegrid <- expand.grid(mtry=recommended.mtry)
rf.1 <- train(growth_2_6 ~ . , data = train_PCA, method = "rf" ,
              ntree = 200, importance = TRUE, trControl = oob_train_control, tuneGri
d = tunegrid)
```

Other Methods Tested

3.1 Collection of Code for models and methods not used for final models

```
### Ridge Reogression
# finding best lambda
i.exp <- seq(10, -5, length = 500)
grid <- 10^i.exp
cv_lambda <- cv.glmnet(model.matrix(growth_2_6~.,prj)[,-1], prj$growth_2_6,
                       family = "gaussian", alpha = 0, lambda = grid,
                       standardize = TRUE, nfolds = 10)
ridge_MSE <- numeric(0)
for (i in folds) {
  train <- model.matrix(growth_2_6~.,prj[-i, ])[,-1]
  train_y <- prj$growth_2_6[-i]
  lambda <- cv_lambda$lambda.1se
  ridge <- glmnet(train, train_y, family = "gaussian", alpha = 0,
                  lambda = lambda, standardize = TRUE)
  test <- model.matrix(growth_2_6~.,prj[i, ])[,-1]
  test_y <- prj$growth_2_6[i]
  ridge_pred <- predict(ridge, test)

  ridge_MSE <- c(ridge_MSE, mean((test_y - ridge_pred)^2))
}

### LASSO
# finding best lambda
i.exp <- seq(10, -5, length = 500)
grid <- 10^i.exp
cv_lambda <- cv.glmnet(model.matrix(growth_2_6~.,prj)[,-1], prj$growth_2_6,
                       family = "gaussian", alpha = 1, lambda = grid,
                       standardize = TRUE, nfolds = 10)
lasso_MSE <- numeric(0)
for (i in folds) {
  train <- model.matrix(growth_2_6~.,prj[-i, ])[,-1]
  train_y <- prj$growth_2_6[-i]
  lambda <- cv_lambda$lambda.1se
  lasso <- glmnet(train, train_y, family = "gaussian", alpha = 1,
                  lambda = lambda, standardize = TRUE)
  test <- model.matrix(growth_2_6~.,prj[i, ])[,-1]
  test_y <- prj$growth_2_6[i]
  lasso_pred <- predict(lasso, test)

  lasso_MSE <- c(lasso_MSE, mean((test_y - lasso_pred)^2))
}
# Coefficients
lasso <- glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$growth_2_6),
                     family = "gaussian", alpha = 1,
                     lambda = grid, standardize = TRUE)

i.exp <- seq(10, -5, length = 500)
grid <- 10^i.exp
```

```r
cv_lambda_lasso <- cv.glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$g
rowth_2_6),
                             family = "gaussian", alpha = 1, lambda = grid,
                             standardize = TRUE, nfolds = 10)
lasso_best <- glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$growth_2_
6),
                     family = "gaussian", alpha = 1,
                     lambda = cv_lambda_lasso$lambda.min, standardize = TRUE)
lasso_coef <- predict(lasso_best,lambda,type = "coefficients")
lasso_coef <- as.numeric(lasso_coef)[-1]

lasso_coef <- which(lasso_coef!=0) #not zero
lasso_coef <- which(abs(lasso_coef)>= 0.025) # greater than 0.025
lasso_coef <- which(abs(lasso_coef)>= 0.05) # greater than 0.05

lasso_coef <- colnames(prj)[lasso_coef]

### Elastic Net
alp <- 0.8
enet <- glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$growth_2_6),
               family = "gaussian", alpha = alp,
               lambda = grid, standardize = TRUE)
plot(enet, xvar = "lambda", label = TRUE)
cv_lambda_enet <- cv.glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$gr
owth_2_6),
                             family = "gaussian", alpha = alp, lambda = grid,
                             standardize = TRUE, nfolds = 10)
enet_best <- glmnet(scale(model.matrix(growth_2_6~.,prj))[,-1], scale(prj$growth_2_
6),
                     family = "gaussian", alpha = alp,
                     lambda = cv_lambda_enet$lambda.1se, standardize = TRUE)
enet_coef <- predict(enet_best,lambda,type = "coefficients")
enet_coef <- as.numeric(enet_coef)[-1]

enet_coef <- which(enet_coef!=0) #not zero
enet_coef <- which(abs(enet_coef)>= 0.1) # greater than 0.1
enet_coef <- which(abs(enet_coef)>= 0.05) # greater than 0.025

enet_coef <- colnames(prj)[enet_coef]

### PCR
pcr_fit <- pcr(growth_2_6~., data = prj, scale = TRUE, validation = "CV")
validationplot(pcr_fit, val.type = 'MSEP')
pcr_MSE <- numeric(0)
for (i in folds) {
  pcr_fit <- pcr(growth_2_6~., data = prj[-i,], scale = TRUE)
  pcr_pred <- predict(pcr_fit, prj[i,], ncomp=50)
  pcr_MSE <- c(pcr_MSE, mean((prj$growth_2_6[i] - pcr_pred)^2))
}

### PLS
pls_fit <- plsr(growth_2_6~., data = prj, scale = TRUE, validation = "CV")
validationplot(pls_fit, val.type = 'MSEP')
pls_MSE <- numeric(0)
for (i in folds) {
  pls_fit <- plsr(growth_2_6~., data = prj[-i,], scale = TRUE)
  pls_pred <- predict(pls_fit, prj[i,], ncomp=50)
  pls_MSE <- c(pls_MSE, mean((prj$growth_2_6[i] - pls_pred)^2))
}

### PCA then KNN
pca_ft <- cbind(pca$scores[,1:100],"growth_2_6" = prj$growth_2_6)
pca_ft <- as.data.frame(pca_ft)
knn_MSE <- numeric(0)
for (i in folds) {
  fit_knn <- knnreg(pca_ft[-i,-ncol(pca_ft)], pca_ft$growth_2_6[-i], k = 10)
  pred_knn <- predict(fit_knn, pca_ft[i,-ncol(pca_ft)])
  knn_MSE <- c(knn_MSE, mean((pca_ft$growth_2_6[i] - pred_knn)^2))
}

View(cbind(pred_knn,pca_ft$growth_2_6[i],pca_ft$growth_2_6[i] - pred_knn))
View(cbind(prj$growth_2_6[i],pca_ft$growth_2_6[i]))

### LASSO then PCA then KNN
lasso_ft <- cbind(prj[,lasso_coef], "growth_2_6" = prj$growth_2_6)

pca_lasso_min <- princomp(lasso_ft[,-ncol(lasso_ft)], cor = TRUE)
var <- pca_lasso_min[["sdev"]]^2
var_prop <- var/sum(var)
plot(cumsum(var_prop))

lasso_pca_ft <- cbind(pca_lasso_min$scores[,1:21],"growth_2_6" = prj$growth_2_6)
lasso_pca_ft <- as.data.frame(lasso_pca_ft)
knn_MSE <- numeric(0)
for (i in folds) {
  fit_knn <- knnreg(lasso_pca_ft[-i,-ncol(lasso_pca_ft)], lasso_pca_ft$growth_2_6[-
i], k = 150)
  pred_knn <- predict(fit_knn, lasso_pca_ft[i,-ncol(lasso_pca_ft)])
  knn_MSE <- c(knn_MSE, mean((lasso_pca_ft$growth_2_6[i] - pred_knn)^2))
}

### LASSO then PCR
lasso_ft <- cbind(prj[,lasso_coef], "growth_2_6" = prj$growth_2_6)
pcr_fit <- pcr(growth_2_6~., data = lasso_ft, scale = TRUE, validation = "CV")
validationplot(pcr_fit, val.type = 'MSEP')
pcr_MSE <- numeric(0)
for (i in folds) {
  pcr_fit <- pcr(growth_2_6~., data = lasso_ft[-i,], scale = TRUE)
  pcr_pred <- predict(pcr_fit, lasso_ft[i,], ncomp=150)
  pcr_MSE <- c(pcr_MSE, mean((lasso_ft$growth_2_6[i] - pcr_pred)^2))
}
View(cbind(pcr_pred, prj$growth_2_6[i]))

### LASSO then KNN
lasso_ft <- cbind(prj[,lasso_coef], "growth_2_6" = prj$growth_2_6)
lasso_ft[,-ncol(lasso_ft)] <- scale(lasso_ft[,-ncol(lasso_ft)])
lasso_knn_MSE <- numeric(0)
for (i in folds) {
  fit_knn <- knnreg(lasso_ft[-i,-ncol(lasso_ft)], lasso_ft$growth_2_6[-i], k = 200)
  pred_knn <- predict(fit_knn, lasso_ft[i,-ncol(lasso_ft)])
  lasso_knn_MSE <- c(lasso_knn_MSE, mean((lasso_ft$growth_2_6[i] - pred_knn)^2))
}

### PLS then KNN
```

```r
pls_ft <- cbind(pls_fit$scores[,1:10],"growth_2_6" = prj$growth_2_6)
pls_ft <- as.data.frame(pls_ft)

knn_MSE <- numeric(0)
for (i in folds) {
  fit_knn <- knnreg(pls_ft[-i,-ncol(pls_ft)], pls_ft$growth_2_6[-i], k = 115)
  pred_knn <- predict(fit_knn, pls_ft[i,-ncol(pls_ft)])
  knn_MSE <- c(knn_MSE, mean((pls_ft$growth_2_6[i] - pred_knn)^2))
}

View(cbind(pred_knn,pls_ft$growth_2_6[i],pls_ft$growth_2_6[i] - pred_knn))
View(cbind(prj$growth_2_6[i],pls_ft$growth_2_6[i]))

### PCA then RF
pca_ft <- cbind(pca$scores[,1:150],"growth_2_6" = prj$growth_2_6)
pca_ft <- as.data.frame(pca_ft)

rf_tree_pca <- randomForest(growth_2_6~., data = pca_ft, ntree = 150, importance = T
RUE)
plot(rf_tree_pca)
# importance
varImpPlot(rf_tree_pca)
rf_tree_pca_imp <- rf_tree_pca$importance
# out of bag method
oob_train_control <- trainControl(method="oob", savePredictions = TRUE)
# rfm with best m
rf_pca <- train(growth_2_6 ~ . , data = pca_ft, method = "rf" ,
            ntree = 100, importance = FALSE, trControl = oob_train_control)
plot(rf_pca)

### LASSO then RF
lasso_ft <- cbind(prj[,lasso_coef], "growth_2_6" = prj$growth_2_6)

rf_tree_lasso <- randomForest(growth_2_6~., data = lasso_ft, ntree = 150, importance
 = TRUE)
plot(rf_tree_lasso)
# importance
varImpPlot(rf_tree_lasso)
rf_tree_lasso_imp <- rf_tree_lasso$importance
# out of bag method
oob_train_control <- trainControl(method="oob", savePredictions = TRUE)
# rfm with best m
rf_lasso <- train(growth_2_6 ~ . , data = lasso_ft, method = "rf" ,
                ntree = 200, importance = FALSE, trControl = oob_train_control)
plot(rf_lasso)

### LASSO + PCA then
lasso_ft <- cbind(prj[,lasso_coef], "growth_2_6" = prj$growth_2_6)

pca_lasso_min <- princomp(lasso_ft[,-ncol(lasso_ft)], cor = TRUE)
var <- pca_lasso_min[["sdev"]]^2
var_prop <- var/sum(var)
plot(cumsum(var_prop))

lasso_pca_ft <- cbind(pca_lasso_min$scores[,],"growth_2_6" = prj$growth_2_6)
lasso_pca_ft <- as.data.frame(lasso_pca_ft)
```

```r
rf_tree_lasso_pca <- randomForest(growth_2_6~., data = lasso_pca_ft, ntree = 150, im
portance = TRUE)
plot(rf_tree_lasso_pca)
# importance
varImpPlot(rf_tree_lasso_pca)
rf_tree_lasso_pca_imp <- rf_tree_lasso_pca$importance
# out of bag method
oob_train_control <- trainControl(method="oob", savePredictions = TRUE)
# rfm with best m
rf_lasso_pca <- train(growth_2_6 ~ . , data = lasso_pca_ft, method = "rf" ,
                  ntree = 200, importance = FALSE, trControl = oob_train_control)
plot(rf_lasso_pca)
```