Final Project - The Gould-en Rule

Stats 101C Lecture 3

Andy Shen, Ethan Allavarpu

Fall 2020

Introduction

The purpose of this regression analysis was to predict the growth percentage of a newly uploaded YouTube video from the 2nd through 6th hour of its publication (`growth_2_6`), which may indicate the virality of a YouTube video (i.e., how quickly it can go viral and "break the Internet," measuring engagement) while also providing a rough guideline for the amount of revenue a creator may receive for a given video. In this analysis, we employ a variety of regression techniques to a variety of attributes that make up a YouTube video.

Methodology

Preprocessing

Data Cleaning We cleaned the data by plotting each predictor variable against `growth_2_6` and examined each univariate plot for associations between predictor variables and the response. We noticed that there existed many outliers or stray points that did not belong in the plot. We systematically removed these points, basing it off personal judgment as to whether the points would not assist in prediction. We do remove highly correlated variables as indicated by a correlation matrix: for predictor pairs with a correlation coefficient over 0.9, we removed one of the two predictors to avoid issues with correlation.

We also transformed multiple variables and added another variable prior to performing predictor selection, as we saw these as potentially useful for data on YouTube growth. The first adjustment we performed was on the `PublishedDate` variable. Initially, the data set had `PublishedDate` as a character variable; we wrote a function to transform this variable into a numeric variable in terms of minutes (since January 1^{st}, 2020 00:00 UTC), a method similar to that described by the `lubridate` package in R (Spinu, 2020). This would allow us to see if there was any relationship between the date and `growth_2_6` because we made the variable continuous to match the perception of time. Not only did we do this with respect to general time since January 1^{st}, 2020 00:00, but we also *added* a variable (`TimeDay`) that has the time of day (in minutes from 00:00 UTC) during which a video was uploaded to YouTube. This could be useful because people may be more likely to watch YouTube videos in the evening (after school or work), so posting nearer to those times allows for greater growth between the second and sixth hours of a video's upload.

The second transformation we made was to the binary variables in the dataset. When looking at the feature descriptions, we noticed that the `low`, `low_mid`, and `mid_high` versions of the four variables were natural ordinal categorical variables (where `high` corresponds to values of 0 for the other three options). We combined these variables into a single factor variable with four levels (0 = `low`, 1 = `low_mid`, 2 = `mid_high`, and 3 = `high`). We did this transformation because bagging and random forests can split along multiple levels of a factor at a node (i.e. 0 and 1 to the left, 2 and 3 to the right) as long as the levels are of a single variable, but if we left it as three separate binary variables, the bagging and random forest models could only split between one level and the three others since it can only look at a single variable for each node (James, Witten, Hastie, & Tibshirani, 2017, p. 313). By combining these variables into a single factor with four levels, we allow the random forest and bagging models greater ability to distinguish between the four levels (such as 2-2 splits) and find nuances for each of these statistics (`Num_Subscribers`, `Num_Views`, `avg_growth`, `count_views`), prompting us to use this coding scheme for these categorical variables.

Predictor Selection In order to refine our subset of predictors, we use the LASSO to select significant predictors. We first fit a LASSO model for a sequence of candidate λ values. From there, we select our optimal value of λ as the one that is one standard deviation above the λ value that resulted in the lowest test MSE. From these parameters, we extract the predicted coefficients in this LASSO model as our predictors for the candidate model.

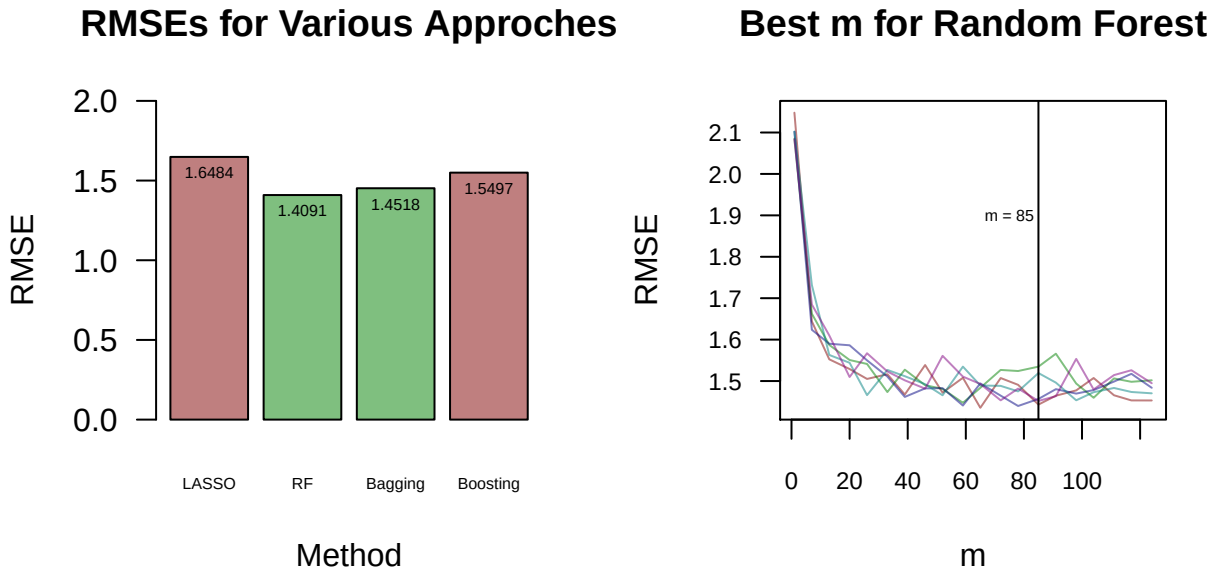
We use LASSO to refine our predictors because this technique shrinks coefficient estimates to 0 and keeps the most important ones. As such, we are only interested in the predictors with nonzero coefficients. LASSO is unrelated to the our candidate model of random forest, and it was only used as a technique to refine the large number of predictors in the data set.

Statistical Model

A vast majority of our models were fit using either bagging or random forest. Our first model used LASSO to select predictors, but we did very little pre-processing and did not remove outliers. This did not result in a very successful model which is why we decided that pre-processing was necessary. We use least-squares as a preliminary model in order to examine whether an inflexible method such as regression would result in a low or high-bias model. Since the Kaggle score was around 1.65, we knew that, despite needing a more flexible model, we would not need something extremely flexible.

We then implemented a mixture of bagging and random forest, adjusting certain parameters at a time, such as the number of trees, their depth, as well as our λ values. The reason we chose bagging and random forest models over methods such as LASSO for final predictions and boosting are because bagging and random forest models produced lower RMSE values (as described in the paragraph below and below plots).

Our best candidate model, `Model 15d`, utilized the random forest approach. After selecting the variables that seemed important, we used our smaller data set to determine the best value of m. This m corresponds to the number of variables the model randomly considers in each node of each decision tree produced. We utilized 5-fold cross-validation to pick the optimal value of m for our final model used on the test data. We created our own cross-validation function to get a better estimate of the test RMSE and tried several potential values for m, ranging from 1 to p, to determine which m produced the lowest RMSE. In the case of `Model 15d`, we chose the best m based on the minimum *median* RMSE (of the five folds). The reason we chose median and not mean is because with only five folds, a single high or low RMSE could significantly impact the average RMSE for a given m, and the median is more resistant to extreme values.



Upon determining the best m, we fit another random forest on 80% of the training data–this time with 500 trees as opposed to 50. We did this to ensure there weren't any final "surprises" when fitting a model with

regards to RMSE. After fitting the random forest for the given m and checking the validation RMSE, we predicted values of `growth_2_6` for the test data set to submit to Kaggle.

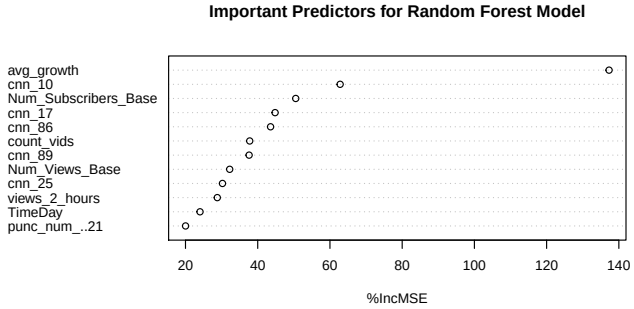
Results

Our best evaluation metric with our primary model (`Model 15d`) on the Kaggle public leaderboard was an RMSE of 1.39594, which beat all four of the threshold models on the public leaderboard. Our secondary model (`Model 15b`) had a Kaggle public leaderboard RMSE of 1.40285, which also beats all four thresholds on the public leaderboard. `Model 15b` only differs from `Model 15d` in the sense that $m = p$ (i.e., a complete bagging approach, p is the number of predictors) as opposed to m equaling the value which produced the lowest median RMSE (i.e., random forest approach).

Conclusions

When comparing the results of our models on the public leaderboard to those on the private leaderboard, we noticed that `Model 15d` ended with a private leaderboard score of _____ while `Model 15b` had a private leaderboard score of _____. FINISH AFTER COMPETITTION

We believe that our primary model (`Model 15d`) performed successfully because it works as an ensemble method, meaning that it combines multiple individual models to get more accurate responses. By using cross-validation for our selection of our best value of m, we limited the potential effect of a random seed showing us an inaccurately good or inaccurately bad RMSE. Moreover, we also used this value of m to fit another random forest model with more trees at the end in order to verify the consistency of our model.



As shown in the variable importance plot of the 12 most influential predictors (out of the 120+ given to the random forest method), `avg_growth`, `Num_Subscribers_Base`, `count_vids`, `Num_Views_Base` and `TimeDay` all are in the top 10% most influential predictors, emphasizing that we made the right decision to combine the binary variables (`avg_growth`, `Num_Subscribers_Base`, `count_vids`, `Num_Views_Base`) and create a new one for the time of day at which the video was uploaded (`TimeDay`).

One thing we believe could have resulted in improved model performance was having greater background knowledge about YouTube and popularity growth as well as the CNN (convoluted neural network) and HOG characteristics for thumbnails. The above plot, which shows the relative importance of the 10 most influential variables in our random forest approach, emphasizes how the CNN vairables played a distinct role in driving the predictions; removing them would have caused the model to perform much worse with respect to MSE (and thus RMSE). As we did not have a great deal of specific knowledge about YouTube, we did not necessarily have an intuitive sense of which predictors might perform better or worse; for data transformation in preprocessing we only changed things and added a predictor that we thought *may* have some influence.

Statement of Contribution

Ethan: Ethan took the lead on the model-building side, primarily testing the bagging and random forest approaches on the data set. He also helped with transforming and creating new features for said model to improve its performance, both in cross-validation as well as the public leaderboard on Kaggle. After model completion, Ethan contributed to the report and slides by helping explain some of the preprocessing steps and adding figures/graphs.

Andy: Andy took the lead on the report and presentation side, helping establish the framework for both the report and slides and explaining most of the thought process of the group for the model-building process. He helped with identifying some key outliers and testing the LASSO, PCA, and boosting methods as potential models. Andy also helped Ethan write some of the code to execute model building and proposed some ideas for altering the model to improve its RMSE.

Code Appendix

```
training <- read.csv("training.csv", stringsAsFactors = FALSE)
dim(training)
any(is.na(training))
var_types <- vapply(training, class, character(1))
names(training)[-which(var_types %in% c("integer", "numeric"))]

dates <- training$PublishedDate
head(dates)
split_dates <- strsplit(dates, "[:/ ]")
head(split_dates)
split_dates <- lapply(split_dates, as.numeric)
years <- function(date) {
  date[3]
}
yrs <- lapply(split_dates, years)
unique(yrs)  # Only 2020
new_time <- function(old_date) {
  old_date <- as.numeric(old_date)
  month <- old_date[1]
  day <- old_date[2]
  hr <- old_date[4]
  minute <- old_date[5]
  month_days <- c(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
  complete_months <- month - 1
  if (complete_months > 0) {
    complete_month_days <- sum(month_days[1:complete_months])
  } else {
    complete_month_days <- 0
  }
  total_days <- complete_month_days + day - 1
  pre_hrs <- total_days * 24
  total_hrs <- pre_hrs + hr
  pre_min <- total_hrs * 60
  final_time <- pre_min + minute
  final_time
}
processed_dates <- vapply(split_dates, new_time, numeric(1))
time_of_day <- function(old_date) {
  hr <- old_date[4]
  minute <- old_date[5]
  tod <- hr * 60 + minute
  tod
}
day_time <- vapply(split_dates, time_of_day, numeric(1))
training$PublishedDate <- processed_dates
training$TimeDay <- day_time

subs <- training[, 248:250]
sub_final <- 3 - 3 * subs[[1]] - 2 * subs[[2]] - subs[[3]]
views <- training[, 251:253]
views_final <- 3 - 3 * views[[1]] - 2 * views[[2]] - views[[3]]
growth <- training[, 254:256]
```

```r
growth_final <- 3 - 3 * growth[[1]] - 2 * growth[[2]] - growth[[3]]
vids <- training[, 257:259]
vid_final <- 3 - 3 * vids[[1]] - 2 * vids[[2]] - vids[[3]]
growth_2_6 <- training$growth_2_6
names(training)[c(1, 248:260)]
training <- training[, -c(1, 248:260)]
training <- data.frame(training, Num_Subscribers_Base = sub_final,
                       Num_Views_Base = views_final, avg_growth = growth_final,
                       count_vids = vid_final, growth_2_6 = growth_2_6)

# See which variables have no variation (useless)
var_sds <- vapply(training, sd, numeric(1))
useless_vars <- names(training)[var_sds == 0]
useless_vars
training <- training[, !(names(training) %in% useless_vars)]

# Remove highly correlated variables
cor_mtx <- round(cor(training[, -ncol(training)]), 2)
library(reshape2)
library(ggplot2)
melted_cor_mtx <- melt(cor_mtx)
cor_vars <- which(abs(cor_mtx) > 0.9, arr.ind = TRUE)
for (i in 1:nrow(cor_vars)) {
  if (cor_vars[i, 1] >= cor_vars[i, 2]) {
    cor_vars[i, ] <- rep(NA, 2)
  }
}

all_na <- function(data) {
  all(is.na(data))
}
unique_cor_vars <- apply(cor_vars, 1, all_na)
cor_vars <- cor_vars[!unique_cor_vars, ]
corr_vars <- unique(rownames(cor_vars))

plot(growth_2_6 ~ ., data = training,
     cex = 0.5, pch = 19, col = rgb(0, 0, 0, 0.1))

library(dplyr)
training <- training[-which(training$views_2_hours > 2000000), ]
training <- training[-which(training$hog_0 > 0.5), ]
training <- training[-which(training$hog_1 > 0.35), ]
training <- training[-which(training$hog_152 > 0.4), ]
training <- training[-which(training$hog_182 > 0.4), ]
training <- training[-which(training$hog_350 > 0.5), ]
training <- training[-which(training$hog_364 > 0.6), ]
training <- training[-which(training$hog_512 > 0.5), ]
training <- training[which(training$hog_522 <= 0.5), ]
training <- training[which(training$hog_584 <= 0.5), ]
training <- training[which(training$hog_643 <= 0.8), ]
training <- training[which(training$hog_649 <= 0.5), ]
training <- training[which(training$hog_658 <= 0.5), ]
training <- training[which(training$hog_705 <= 0.5), ]
```

```r
training <- training[which(training$hog_724 <= 0.5), ]
training <- training[which(training$hog_774 <= 0.5), ]
training <- training[which(training$hog_818 <= 0.4), ]
training <- training[which(training$hog_828 <= 0.6), ]
training <- training[which(training$hog_831 <= 0.5), ]
training <- training[which(training$hog_832 <= 0.6), ]
training <- training[which(training$hog_855 <= 0.5), ]
training <- training[which(training$cnn_0 == 0), ]
training <- training[which(training$cnn_36 == 0), ]
training <- training[which(training$cnn_65 == 0), ]
training <- training[which(training$punc_num_..18 == 0), ]
training <- training[which(training$punc_num_..21 <= 1.5), ]
training <- training[which(training$punc_num_..26 == 0), ]
training <- training[which(training$punc_num_..27 == 0), ]

new_var_sd <- vapply(training, sd, numeric(1))
useless_vars <- names(training)[new_var_sd == 0]
useless_vars
training <- training[, !(names(training) %in% useless_vars)]

factor_vars <- training[, 230:233]
factor_vars <- data.frame(lapply(factor_vars, factor))
training[, 230:233] <- factor_vars
training <- training[, !(names(training) %in% corr_vars)]

plot(growth_2_6 ~ ., data = training,
     cex = 0.5, pch = 19, col = rgb(0, 0, 0, 0.1))

val_mses <- numeric(10)
set.seed(1234)
library(glmnet)
for (i in seq_len(10)) {
  data_split <- sample(nrow(training), nrow(training) * 0.8)
  train <- training[data_split, ]
  validation <- training[-data_split, ]
  train_x <- model.matrix(growth_2_6 ~ ., data = train)[, -1]
  train_y <- train$growth_2_6
  test_x <- model.matrix(growth_2_6 ~ ., data = validation)[, -1]
  lambda_grid <- 10^(seq(from = 10, to = -2, length.out = 100))
  salary_lasso <- glmnet(train_x, train_y, family = "gaussian",
                         alpha = 1, lambda = lambda_grid, standardize = TRUE)
  salary_lasso_cv <- cv.glmnet(train_x, train_y, family = "gaussian", alpha = 1,
                               lambda = lambda_grid, standardize = TRUE,
                               nfolds = 10)
  lambda_1se <- salary_lasso_cv$lambda.min
  lasso_test_preds <- predict(salary_lasso, newx = test_x, s = lambda_1se)
  val_mses[i] <- mean((lasso_test_preds - validation$growth_2_6)^2)
}
mean(val_mses)
hist(val_mses)
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, ]
validation <- training[-data_split, ]
```

```r
train_x <- model.matrix(growth_2_6 ~ ., data = train)[, -1]
train_y <- train$growth_2_6
test_x <- model.matrix(growth_2_6 ~ ., data = validation)[, -1]
lambda_grid <- 10^(seq(from = 10, to = -2, length.out = 100))
salary_lasso <- glmnet(train_x, train_y, family = "gaussian",
                       alpha = 1, lambda = lambda_grid, standardize = TRUE)
salary_lasso_cv <- cv.glmnet(train_x, train_y, family = "gaussian", alpha = 1,
                             lambda = lambda_grid, standardize = TRUE,
                             nfolds = 10)
lambda_1se <- salary_lasso_cv$lambda.min
lasso_coefs <- predict(salary_lasso, newx = test_x,
                       s = lambda_1se, type = "coefficients")[, 1]
keep_vars <- names(lasso_coefs)[lasso_coefs != 0][-1]

keep_vars <- c(keep_vars[1:120], names(training)[163:166])

library(randomForest)
set.seed(100)
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, ]
validation <- training[-data_split, ]
train_bag <- train[, c(keep_vars, "growth_2_6")]
# -1 from ncol() because one column is the response variable
p <- ncol(train_bag) - 1 # Total number of predictors = 59
m_vals <- floor(seq(from = 1, to = p, length.out = 20))

create_best_m <- function(m_val, train_data = training, split = 0.8, folds = 5,
                          num_tree = 25, keep_vars = names(train_data)) {
  data_split <- sample(nrow(train_data), nrow(train_data))
  rmses <- numeric(folds)
  for (i in seq_len(folds)) {
    d_split <- data_split[((nrow(train_data) / folds)*(i - 1) + 1):((nrow(train_data) / folds) * i)]
    train <- train_data[-d_split, keep_vars]
    validation <- train_data[d_split, ]
    bag_tree <- randomForest(growth_2_6 ~ ., data = train,
                             mtry = m_val, ntree = num_tree)
    bag_preds <- predict(bag_tree, newdata = validation)
    bag_mse <- mean((validation$growth_2_6 - bag_preds)^2)
    rmses[i] <- sqrt(bag_mse)
  }
  rmses
}
set.seed(9738-16)
rmses <- vapply(m_vals, create_best_m, numeric(5),
                train_data = training,
                keep_vars = c(keep_vars, "growth_2_6"),
                num_tree = 50)


plot(m_vals, rmses[1, ], type = "l", ylim = range(rmses))
for (i in 2:5) {
  lines(m_vals, rmses[i, ], col = i)
}
```

```r
set.seed(129)
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, c(keep_vars, "growth_2_6")]
validation <- training[-data_split, ]
rf_tree <- randomForest(growth_2_6 ~ ., data = train,
                        mtry = m_vals[which.min(apply(rmses, 2, median))],
                        ntree = 500, importance = TRUE)
rf_preds <- predict(rf_tree, newdata = validation)
rf_mse <- mean((validation$growth_2_6 - rf_preds)^2)
rmse <- sqrt(rf_mse)
rmse

set.seed(0)
data_split <- sample(nrow(training), .8 * nrow(training))
train <- training[data_split, c(keep_vars, "growth_2_6")]
validation <- training[-data_split, ]
bag_tree <- randomForest(growth_2_6 ~ ., data = train,
                         mtry = p,
                         ntree = 500)
bag_preds <- predict(bag_tree, newdata = validation)
bag_mse <- mean((validation$growth_2_6 - bag_preds)^2)
bag_rmse <- sqrt(bag_mse)
bag_rmse

library(gbm)
set.seed(1265)
lambdas <- 10^seq(from = -10, to = 0, length.out = 100)
boost_rmses <- rep(NA, length(lambdas))
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, ]
validation <- training[-data_split, ]
train_bag <- train[, c(keep_vars, "growth_2_6")]
# -1 from ncol() because one column is the response variable
p <- ncol(train_bag) - 1 # Total number of predictors = 59
for(i in seq_along(boost_rmses)) {
  boosted <- gbm(growth_2_6 ~ ., data = train_bag, distribution = "gaussian",
                 n.trees = 1000, shrinkage = lambdas[i], verbose = FALSE)
  boost_preds <- predict(boosted, newdata = validation)
  boost_mse <- mean((validation$growth_2_6 - boost_preds)^2)
  boost_rmses[i] <- sqrt(boost_mse)
}
best_lam <- lambdas[which.min(boost_rmses)]
best_lam

set.seed(126)
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, c(keep_vars, "growth_2_6")]
validation <- training[-data_split, ]
boosted <- gbm(growth_2_6 ~ ., data = train_bag, distribution = "gaussian",
               n.trees = 500, shrinkage = best_lam, verbose = FALSE)
boost_preds <- predict(boosted, newdata = validation)
boost_mse <- mean((validation$growth_2_6 - boost_preds)^2)
boost_rmse <- sqrt(boost_mse)
```

```r
boost_rmse

all_rmses <- c(sqrt(mean(val_mses)), rmse, bag_rmse, boost_rmse)
names(all_rmses) <- c("LASSO", "Random Forest", "Bagging", "Boosting")
barplot(all_rmses, ylim = c(0, 2), las = 1,
        col = c(rgb(0.5, 0, 0, 0.5), rgb(0, 0.5, 0, 0.5),
                rgb(0, 0.5, 0, 0.5), rgb(0.5, 0, 0, 0.5)),
        xlab = "Method", ylab = "RMSE", main = "RMSEs for Various Approches")
text(c(.7, 1.9, 3.1, 4.3), all_rmses - 0.1, labels = round(all_rmses, 4))

varImpPlot(rf_tree, type = 1, scale = TRUE, n.var = 12, main = "Important Predictors for Random Forest

test <- read.csv("test.csv")
dates <- test$PublishedDate
split_dates <- strsplit(dates, "[:/ ]")
split_dates <- lapply(split_dates, as.numeric)
years <- function(date) {
  date[3]
}
yrs <- lapply(split_dates, years)
unique(yrs)
processed_dates <- vapply(split_dates, new_time, numeric(1))
test$PublishedDate <- processed_dates
day_time <- vapply(split_dates, time_of_day, numeric(1))
test$TimeDay <- day_time
subs <- test[, 248:250]
sub_final <- 3 - 3 * subs[[1]] - 2 * subs[[2]] - subs[[3]]
views <- test[, 251:253]
views_final <- 3 - 3 * views[[1]] - 2 * views[[2]] - views[[3]]
growth <- test[, 254:256]
growth_final <- 3 - 3 * growth[[1]] - 2 * growth[[2]] - growth[[3]]
vids <- test[, 257:259]
vid_final <- 3 - 3 * vids[[1]] - 2 * vids[[2]] - vids[[3]]
id <- test$id
test <- test[, -c(1, 248:259)]
test <- data.frame(test, Num_Subscribers_Base = sub_final,
                   Num_Views_Base = views_final, avg_growth = growth_final,
                   count_vids = vid_final)
factor_vars <- test[, 248:251]
factor_vars <- data.frame(lapply(factor_vars, factor))
test[, 248:251] <- factor_vars
rf_test_preds <- predict(rf_tree, newdata = test)
test_csv <- data.frame(id = id, growth_2_6 = rf_test_preds)
# write.csv(test_csv, "model15dpredictions.csv", row.names = FALSE)
bag_test_preds <- predict(bag_tree, newdata = test)
test_csv_bag <- data.frame(id = id, growth_2_6 = bag_test_preds)
# write.csv(test_csv_bag, "model15bpredictions.csv", row.names = FALSE)
```

References

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning with applications in r*. New York, NY: Springer.

Spinu, V. (2020). *Lubridate v1.7.9.2*. Retrieved from https://www.rdocumentation.org/packages/lubridate/versions/1.7.9.2