# 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 plotted each predictor against `growth_2_6` and looked for associations between predictors and the response. There existed outliers or points that did not belong in the plot; we systematically removed these points, basing it on personal judgment on how "far away" the points seemed. We also removed highly correlated variables: for predictor pairs with a correlation coefficient over 0.9, we removed one of the two predictors to avoid issues with correlation. Lastly, we removed variables that had no standard deviation because they would be useless in a model and would only slow down computing time.

We transformed variables and added one more prior to performing predictor selection, as we saw them as potentially useful for data on YouTube growth. Initially, `PublishedDate` was a character variable; we transformed it into a numeric variable in terms of minutes (since January $1^{st}$, 2020 00:00 UTC), a method similar to that described in the `lubridate` package (Spinu, 2020). This would help us see any relationship between the date and `growth_2_6` on a continuous scale. Not only did we do this for general time since January $1^{st}$, 2020 00:00 UTC, but we also *added* a variable (`TimeDay`) which has the time (in minutes from 00:00 UTC) at 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/work), so uploading closer to those times allows for greater growth between the second and sixth hours since 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 levels (where `high` corresponds to values of 0 for the other three options). We combined these variables into a single factor with four levels (0 = `low`, 1 = `low_mid`, 2 = `mid_high`, and 3 = `high`). We did this 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) if the levels are from a single variable; if we left three separate binary variables, bagging and random forests could only split between one level and the three others since it can only split on a single variable at 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) to find nuances for each of these statistics (`Num_Subscribers`, `Num_Views`, `avg_growth`, `count_views`). Specifically for random forests, by combining these binary variables into a single factor, we know all four levels will be considered if the factor is chosen as part of the subset of $m$ predictors at each node–if left as three separate binary variables, it would be possible for the random forest to select only one of the three in its subset, thus not completely capturing the differences across all four levels.

**Predictor Selection** To refine our subset of predictors, we use the LASSO to select significant predictors. We fit a LASSO model for a sequence of candidate $\lambda$ values, from $10^{-5}$ to $10^{5}$: larger values of $\lambda$ lead to greater shrinkage and more selection, while smaller values of $\lambda$ have less shrinkage and are more similar to OLS. From there, we select our optimal value of $\lambda = 10^{-2} = 0.01$ as the one with the lowest cross-validation MSE. From this value of $\lambda$, we extract the features with non-zero predicted coefficients in this LASSO model as our predictors for the candidate model.

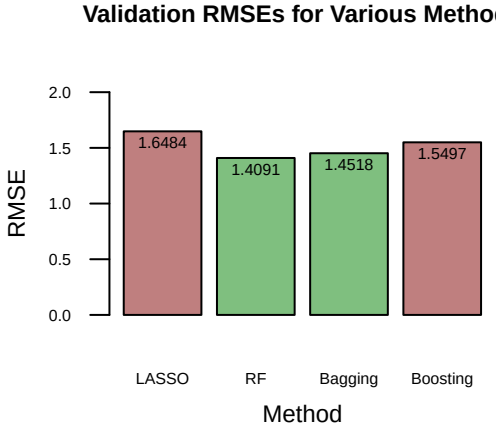
We use LASSO to refine our predictors because this technique shrinks coefficient estimates to 0 and keeps the most important ones. As such, we are only interested in the predictors with nonzero coefficients. LASSO is unrelated to the our candidate model of random forest, and it was only used as a technique to refine the large number of predictors in the data set. The reason we chose LASSO over PCA is because PCA does not work well with categorical variables, while LASSO establishes dummy variables: if any of the dummy variables were selected, we kept the entire factor with all four levels.

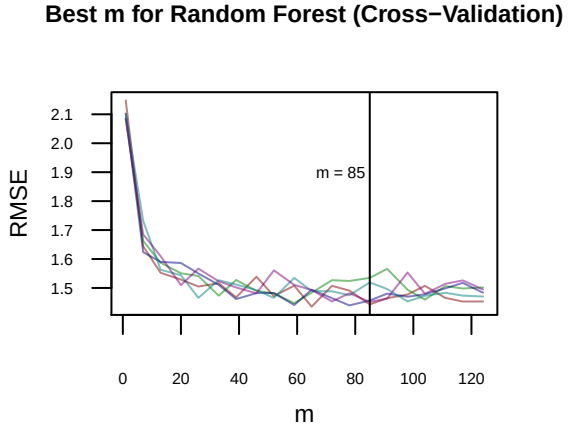
## Statistical Model

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 model with extremely high bias. Since the Kaggle score was around 1.65, we knew that, though we would need a more flexible model, it need not be 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 $\lambda$ 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 $m$ (85), 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.

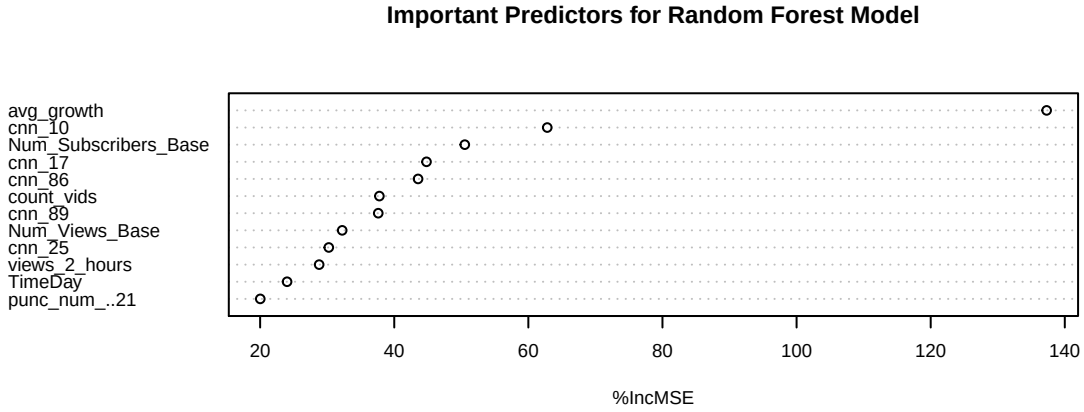
## Results

Our best evaluation metric with our primary model (`Model 15d`) on the Kaggle public leaderboard was an RMSE of 1.39753, which beats all four threshold models on the public leaderboard. Our secondary model (`Model 15e`) had a Kaggle public leaderboard RMSE of 1.40637, which also beats all four thresholds. `Model 15e` differs from `Model 15d` only in that $m = p$ (i.e., bagging, $p =$ the number of predictors) as opposed to $m$ equaling the value which produced the lowest median RMSE (i.e., random forest).

## Conclusions

For the private Kaggle leaderboard, `Model 15d` had a private score of 1.41019, while `Model 15e` had a private score of 1.41321–both models beat all four thresholds on the private leaderboard as well.

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 rather than relying on a single decision tree. 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 used this value of $m$ to fit another random forest model with more trees at the end to verify the consistency of our model. The reason random forest (`15d`) performed better than bagging (`15e`) was that it limits the bias that bagging introduces by randomly selecting only a subset variables from which to split at a given node, providing for trees that are slightly more independent from one another.




As shown in the variable importance plot of the 12 most influential predictors (from the 124 given to our random forest), `avg_growth`, `Num_Subscribers_Base`, `count_vids`, `Num_Views_Base` and `TimeDay` are all in the top 10% for influential predictors ($1^{\text{st}}$, $3^{\text{rd}}$, $6^{\text{th}}$, $8^{\text{th}}$, and $11^{\text{th}}$, respectively). We made a good 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 of an upload (`TimeDay`); our feature transformation and creation proved useful in aiding our model's strong performance on the public leaderboard.

One thing we believe could have resulted in improved model performance was having greater knowledge about the CNN and HOG characteristics for thumbnails and combining these predictors in a more efficient manner. The above plot, which shows the relative importance of the 12 most influential variables in our random forest, emphasizes how the CNN variables played a distinct role in driving the predictions; removing them would have caused the model to perform distinctly worse with respect to MSE (and thus RMSE). One potential method we may explore in the future is PCA (principal component analysis) to combine the numerous CNN and HOG variables into only a few (i.e. dimension reduction) to better incorporate these variables in our model and improve the efficiency of our code.

## 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
# Time since 01/01/2020 0:00 UTC
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 since 0:00 UTC (day)
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

# Combine binary variables into factors with four levels
subs <- training[, 248:250]
```

```r
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]
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))

# Remove points which seemed like outliers / distant
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), ]
```

```r
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$hog_705 <= 0.5), ]
training <- training[which(training$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), ]

# Remove variables that have no standard deviation (variation) after removing outliers
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)]

# Convert necessarily variables into factors
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))

# Use LASSO to predict as well as perform feature selection
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$growth_2_6
  test_x <- model.matrix(growth_2_6 ~ ., data = validation)[, -1]
  lambda_grid <- 10^(seq(from = 5, to = -5, length.out = 101))
  growth_lasso <- glmnet(train_x, train_y, family = "gaussian",
                         alpha = 1, lambda = lambda_grid, standardize = TRUE)
  growth_lasso_cv <- cv.glmnet(train_x, train_y, family = "gaussian", alpha = 1,
                               lambda = lambda_grid, standardize = TRUE,
                               nfolds = 10)
  lambda_best <- growth_lasso_cv$lambda.min
```

```
  lasso_test_preds <- predict(growth_lasso, newx = test_x, s = lambda_best)
  val_mses[i] <- mean((lasso_test_preds - validation$growth_2_6)^2)
}
mean(val_mses)
hist(val_mses)
data_split <- sample(nrow(training), nrow(training) * 0.8)
train <- training[data_split, ]
validation <- training[-data_split, ]
train_x <- model.matrix(growth_2_6 ~ ., data = train)[, -1]
train_y <- train$growth_2_6
test_x <- model.matrix(growth_2_6 ~ ., data = validation)[, -1]
lambda_grid <- 10^(seq(from = 5, to = -5, length.out = 101))
growth_lasso <- glmnet(train_x, train_y, family = "gaussian",
                       alpha = 1, lambda = lambda_grid, standardize = TRUE)
growth_lasso_cv <- cv.glmnet(train_x, train_y, family = "gaussian", alpha = 1,
                             lambda = lambda_grid, standardize = TRUE,
                             nfolds = 10)
lambda_best <- growth_lasso_cv$lambda.min
lasso_coefs <- predict(growth_lasso, newx = test_x,
                       s = lambda_best, 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
m_vals <- floor(seq(from = 1, to = p, length.out = 20))

# CV function to see best m for random forest
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[seq(from = ((nrow(train_data) / folds) * (i - 1) + 1),
                              to = ((nrow(train_data) / folds) * i), by = 1)]
    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
}
```

```r
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 vs. RMSEs for random forests
col_scheme <- c(rgb(0.5, 0, 0, 0.5),
                rgb(0, 0.5, 0, 0.5),
                rgb(0, 0, 0.5, 0.5),
                rgb(0, 0.5, 0.5, 0.5),
                rgb(0.5, 0, 0.5, 0.5))
plot(m_vals, rmses[1, ], type = "l", ylim = range(rmses),
     col = col_scheme[1],
     xlab = "m", ylab = "RMSE", las = 1,
     cex.axis = 0.5, cex.main = 0.5, cex.lab = 0.5)
for (i in 2:5) {
  lines(m_vals, rmses[i, ], col = col_scheme[i])
}
abline(v = m_vals[which.min(apply(rmses, 2, median))])
text(m_vals[which.min(apply(rmses, 2, median))] - 10,
     1.9,
     labels = paste("m = ",
                    m_vals[which.min(apply(rmses, 2, median))],
                    sep = ""), cex = 0.5)

# Choose value of m with lowest median RMSE
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

# Complete bagging approach (m = p)
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, importance = TRUE)
bag_preds <- predict(bag_tree, newdata = validation)
bag_mse <- mean((validation$growth_2_6 - bag_preds)^2)
bag_rmse <- sqrt(bag_mse)
bag_rmse

# Boosting method
```

```r
## Find best value of lambda
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
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

## Use best value of lambda and test on validation RMSE
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)
boost_rmse

# Plot RMSEs of various methods
all_rmses <- c(sqrt(mean(val_mses)), rmse, bag_rmse, boost_rmse)
names(all_rmses) <- c("LASSO", "RF", "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",
        cex.names = 0.5, las = 1, cex.main = 0.5, cex.lab = .5, cex.axis = .5)
text(c(.7, 1.9, 3.1, 4.3), all_rmses - 0.1, labels = round(all_rmses, 4),
     cex = .5)

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

# Transform the test dataset accordingly and write the prediction .csv files
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, "model15epredictions.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