# STATS 101C - Final Report

Red Team 1 (Darren Tsang, Yonatan Khalil), Lecture 4

Produced on Monday, Dec. 14 2020 @ 09:14:28 PM

## 1 Introduction

YouTube, an online video-sharing platform, has grown significantly since its 2005 founding and later purchase by Google in 2006. The site has catered to many viewers by supplying videos ranging from cute kittens to news to tutoring. Video creators often rely on analytics to help guide them towards a successful video with a high amount of views. One such analytic is the growth rate.

For this Kaggle project, we are given a training dataset containing 7242 observations, where each observation represents a video. We are also given 259 predictors associated with each of the videos; these predictors are a combination of continuous and discrete. We are open to using methods we've learned in this course (eg. random forest, boosting) or even methods beyond the scope of this course. Our goal is to use the provided training dataset to train a model to predict *growth_2_6*, the percent change of a video's views between the second and sixth hour from posting. Then, we put our trained model to the test on the testing dataset.

The evaluation metric used for our model's performance is the root mean squared error (RMSE), which is shown in Equation 1. $g_i$ represents the actual growth percentage, $\hat{g}_i$ represents the growth percentage predicted by our model, and $n$ is the total number of predictions made. Please note that unlike the midterm, this time there are 4 models (Model 1, Model 2, Model 3, Model 4) created by the course staff to compare our model's performance with. In other words, we are not in competition with other teams.

$$RMSE = \left( \frac{\sum_{i=1}^{n} (g_i - \hat{g}_i)^2}{n} \right)^{1/2} \tag{1}$$

When we are given any dataset, it is always a good to perform some exploratory data analysis (EDA). One relatively easy way to gain an understanding of the data quickly is through the use of plots. Figure 1 is a plot we created using the training dataset during our EDA. Just from this simple plot, we can see that videos that are long do not have high *growth_2_6* values.

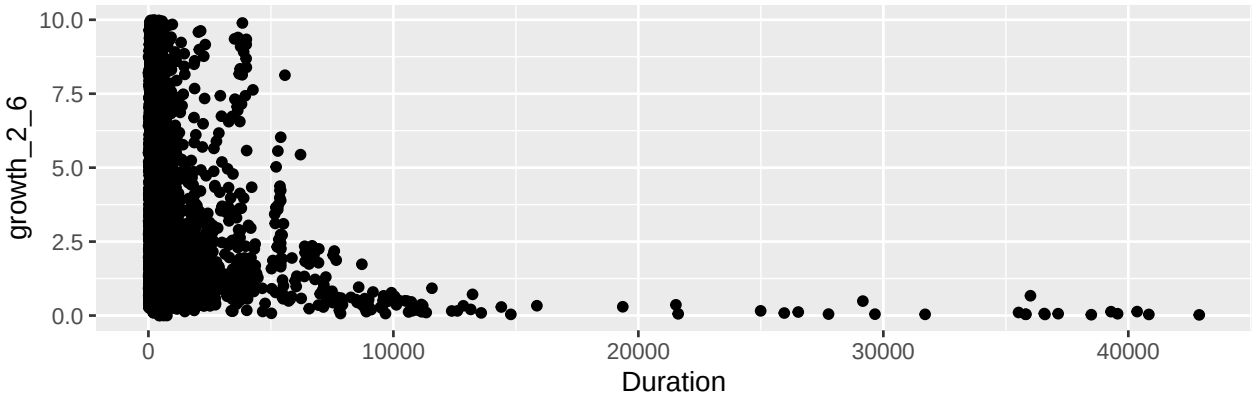



Figure 1: Duration vs growth_2_6 Plot

# 2 Methodologies

Before we cleaned our dataset in any way, we tried bagging on the whole, uncleaned training dataset. Bagging seemed like it would be a great method since it was able to handle both categorical and discrete predictors. To our surprise, we obtained a RMSE of 1.45502 on the public leaderboard, which was already good enough to beat Model 3. Thus, we used bagging as a starting point because we saw that using a very naive approach already yielded us with impressive results. We felt that with a few modifications to our initial approach, we would be able to use bagging to beat Model 4.

## 2.1 Prepocessing

First, we removed the *id* column completely since the *id* was just a unique identifier for each video that was assigned arbitrarily. Thus, we did not believe *id* should be included in any model we created.

Second, we used the *PublishedDate* column to engineer 3 new variables: *month*, *day*, and *min_of_day*. For example, if the *PublishedDate* was "4/17/2020 10:38", the *month* would be 4, *day* would be 17, and *min_of_day* would be $(10 * 60) + 38 = 638$. We decided not to bother with the year because the year for all videos were 2020. This was done because it felt natural that time would have an effect on *growth_2_6*.

Third, we removed columns that summed up to 0 because we assumed that if an entire column summed to 0, the values for that column would be all 0s. If all values of a column were the same, there would be no predictive power.

Fourth and lastly, we used the find_pred() function Darren created during the midterm to help us remove multicollinearity between our predictors. In short, find_pred() takes in a potential list of predictors and returns a subset of that list such that no pair of predictors had a correlation value higher than a threshold set by us.

The final list of predictors we used are shown in the Appendix.

## 2.2 Statistical Model

A decision tree is a relatively simple algorithm that asks a series of questions to determine the outcome. Of course, a decision tree on its own isn't very powerful, which is where bagging comes in. Bagging is an extension of a decision tree; in bagging, you are building multiple decision tress using bootstrap samples of your dataset.

Then, to make a new prediction, you run your new observations through all of your decision trees and take the average outcome (it's slightly different if you are doing a classification problem). This is shown mathematically in Equation 2. Please note that $B$ is the total number of decision trees, $\hat{T}_b$ is the $b^{th}$ decision tree, and $x_i$ are the predictors associated with the $i^{th}$ observation.

$$\hat{y}_{final}(x_i) = \frac{\sum_{b=1}^{B} \hat{T}_b(x_i)}{B} \tag{2}$$

After bagging, we also experimented with random forests. The idea of a random forest is very, very similar to bagging. In bagging, you are using all $p$ predictors when determining how to make a split at each node. On the other hand, in a random forest, you are only using a random sample of $m$ predictors to determine how to make a split, where $m < p$. Random forest does an even better job in reducing variance than bagging. This is because every predictor gets their time to "shine" because the dominant predictors will not be as dominating when compared to bagging.

We discuss our results when using bagging and random forest in the following section.

# 3 Results

| | Training RMSE | Public RMSE | Private RMSE |
|---|---|---|---|
| Random Forest ($m = 165$) | 0.579211 | 1.41278 | 1.39849 |
| Bagging | 0.577314 | 1.41472 | 1.40174 |

Table 1: Summary of our best results.

In Table 1, we have shown the training, public, and private RMSE for the 2 models we chose on Kaggle for our final score. Both of these models were successful, as they beat Model 1, Model 2, Model 3, Model 4 on both the public and private leaderboards.

For both of these models, we:

- used our cleaned dataset for training, as described in *Section 2.1: Preprocessing*,
- set 0.7 as the threshold for our find_pred() function,
- used the recommended minimum number of terminal nodes for regression (5),
- and built 500 trees.

We decided to use 0.7 because after testing out values ranging from 0.3 to 0.8, we saw that 0.7 was best. We went with 500 trees because of it's combination of good performance and acceptable computational time. Lastly, we decided to simply stick with 5 as the minimum number of terminal nodes because the performance was sufficient. Through all of the trial and error, we obtained RMSEs on the public leaderboard ranging from 2.59195 to 1.41278.

As for random forest specifically, we tried $m$ values from a small range of values (162 to 170) because just trying out one $m$ value took a very, very long time. The range was chosen because they were close to $m = 171$, which was the value used for bagging.

# 4 Conclusions

As seen in Table 1, our top two models on the private leaderboard had a RMSE of 1.39849 and 1.40174, which was obtained using random forest and bagging respectively.

We believe that our two models worked well because of the preprocessing performed on our data. This belief is strongly supported empirically when we compare the RMSE when using bagging on the uncleaned dataset vs cleaned dataset (1.45502 vs 1.41472). Lastly, we believe our models were very generalizable because we were not strictly aiming for a low RMSE on the public leaderboard, which prevented overfitting to the training dataset. In fact, this is something we are proud of, as we jumped up 13 spots once the private leaderboard was released.

On the other hand, there are a couple of ways we could have improved our performance. For example, we did not deal with outliers when cleaning our data. If we were to remove outliers, our performance would most likely improve because our model would not be learning from extreme, non-typical values. Also, we could have further explored the idea of using random forest by trying out more values of $m$. One last thing that we could have also tried was creating training and validation datasets. This would have allowed us to evaluate our models on new, unseen data. We would be able to see just how much off our predictions were or if there were any particular areas our models were struggling in (since we would also have the true *growth_2_6* in the validation dataset).

Overall though, we are satisfied with our results, but also understand that our model could be further improved if we were given more time.

**Note: All of our code is located in the Appendix with well-written documentation. The model shown in the code is for bagging, but we have marked where to make changes for random forest.**

# Appendix

### Load in necessary libraries

```
library(tidyverse) # for cleaning data
library(randomForest) # for creating our model
```

### find_pred() function

```
find_pred <- function(df, check, correlation){
 # include all initially
 include <- rep(T, length(check))
 for (i in seq(1, length(check))){
   cur_col = check[i]
   if(include[i]){
     # check which columns have higher correlation than cur_col
     a <- (abs(unname(df[, cur_col])) > correlation)
     # check which columns in a also are in check
     in_both <- intersect(check, names(df[a, cur_col]))
     if(!is.null(in_both)){
       # get check's position that contains each element of in_both
       pos <- match(in_both, check)
       for (j in pos){
         if(i < j){
           # set to false (not including)
           include[j] = F
         }
       }
     }
   }
 }

 return(check[include])
}
```

### Load in datasets and look at how many observations in each

```
training <- read.csv("stats101c-lec4-final-competition/training.csv")
dim(training)
```

```
## [1] 7242  260
```

```
testing <- read.csv("stats101c-lec4-final-competition/test.csv")
dim(testing)
```

```
## [1] 3105  259
```

**Data Cleaning**

```
# remove id variable, as they are assigned arbitrarily
new_training <- training[, -1]
new_testing <- testing[, -1]
```

```
# separate the Date column into the month, day, year, hour, minute
new_training <- separate(new_training, 1, c("month", "day", "year", "hour", "minute"))
new_testing <- separate(new_testing, 1, c("month", "day", "year", "hour", "minute"))
```

```
# converts the month, day, year, hour, minute to numeric
new_training$month <- as.numeric(new_training$month)
new_training$day <- as.numeric(new_training$day)
new_training$year <- as.numeric(new_training$year)
new_training$hour <- as.numeric(new_training$hour)
new_training$minute <- as.numeric(new_training$minute)

# same as above, but this time for new_testing
new_testing$month <- as.numeric(new_testing$month)
new_testing$day <- as.numeric(new_testing$day)
new_testing$year <- as.numeric(new_testing$year)
new_testing$hour <- as.numeric(new_testing$hour)
new_testing$minute <- as.numeric(new_testing$minute)
```

```
# calculate min_of_day
new_training <- cbind("min_of_day" = new_training$hour*60 + new_training$minute,
                      new_training)
new_testing <- cbind("min_of_day" = new_testing$hour*60 + new_testing$minute,
                     new_testing)
```

```
# remove hour and minute columns because we created min_of_day that incorporates both of them
# remove year column because they are all 2020
new_training <- new_training[, !(names(new_training) %in% c("hour", "minute", "year"))]
new_testing <- new_testing[, !(names(new_testing) %in% c("hour", "minute", "year"))]
```

```
# find columns that are all 0 in training dataset
all_zero <- (which(colSums(new_training) == 0))

# from both training and testing datset, remove columns that are all 0
new_training <- new_training[, -all_zero]
new_testing <- new_testing[, -all_zero]
```

```
dim(new_training)
```

```
## [1] 7242  249
```

```
dim(new_testing)
```

```
## [1] 3105  248
```

**Choosing which predictors to use by removing multicollinearity**

```
potential_predictors <- names(sort(abs(cor(new_training)[, "growth_2_6"]), decreasing = T)[-1])
length(potential_predictors)
```

```
## [1] 248
```

```
final_predictors <- find_pred(cor(new_training), potential_predictors, .7)
length(final_predictors)
```

```
## [1] 171
```

```
print(final_predictors)
```

```
##   [1] "cnn_17"                        "Num_Views_Base_mid_high"
##   [3] "cnn_89"                        "avg_growth_low"
##   [5] "num_chars"                     "avg_growth_low_mid"
##   [7] "count_vids_mid_high"           "doc2vec_17"
##   [9] "Num_Views_Base_low"            "Num_Subscribers_Base_mid_high"
##  [11] "num_uppercase_chars"           "Num_Views_Base_low_mid"
##  [13] "Num_Subscribers_Base_low_mid"  "cnn_10"
##  [15] "Num_Subscribers_Base_low"      "avg_growth_mid_high"
##  [17] "doc2vec_10"                    "num_stopwords"
##  [19] "Duration"                      "punc_num_..1"
##  [21] "punc_num_..28"                 "cnn_86"
##  [23] "doc2vec_4"                     "doc2vec_15"
##  [25] "num_uppercase_words"           "hog_704"
##  [27] "doc2vec_11"                    "count_vids_low"
##  [29] "punc_num_..7"                  "hog_705"
##  [31] "views_2_hours"                 "count_vids_low_mid"
##  [33] "punc_num_..3"                  "doc2vec_18"
##  [35] "pct_nonzero_pixels"            "hog_673"
##  [37] "punc_num_..8"                  "hog_702"
##  [39] "hog_855"                       "hog_738"
##  [41] "doc2vec_0"                     "hog_453"
##  [43] "hog_703"                       "hog_116"
##  [45] "hog_666"                       "hog_452"
##  [47] "hog_746"                       "punc_num_..10"
##  [49] "doc2vec_1"                     "hog_858"
##  [51] "hog_640"                       "hog_641"
##  [53] "hog_782"                       "hog_108"
##  [55] "doc2vec_5"                     "hog_716"
##  [57] "hog_378"                       "hog_697"
##  [59] "hog_674"                       "hog_412"
##  [61] "max_green"                     "hog_454"
##  [63] "hog_859"                       "hog_386"
##  [65] "hog_376"                       "doc2vec_13"
##  [67] "hog_659"                       "hog_860"
##  [69] "hog_78"                        "month"
##  [71] "hog_828"                       "hog_125"
##  [73] "hog_863"                       "hog_241"
```

```
##  [75] "hog_825"                  "hog_13"
##  [77] "hog_166"                  "hog_105"
##  [79] "hog_829"                  "hog_287"
##  [81] "hog_0"                    "hog_831"
##  [83] "hog_844"                  "hog_204"
##  [85] "hog_668"                  "hog_242"
##  [87] "hog_791"                  "cnn_88"
##  [89] "hog_655"                  "hog_11"
##  [91] "hog_522"                  "hog_368"
##  [93] "hog_832"                  "hog_295"
##  [95] "hog_1"                    "hog_774"
##  [97] "doc2vec_9"                "hog_815"
##  [99] "hog_857"                  "hog_698"
## [101] "hog_856"                  "hog_106"
## [103] "punc_num_..24"            "hog_827"
## [105] "hog_492"                  "hog_849"
## [107] "hog_155"                  "punc_num_..14"
## [109] "hog_350"                  "hog_117"
## [111] "hog_195"                  "hog_259"
## [113] "hog_21"                   "hog_669"
## [115] "hog_643"                  "hog_783"
## [117] "max_blue"                 "hog_165"
## [119] "hog_658"                  "hog_156"
## [121] "hog_413"                  "hog_316"
## [123] "hog_442"                  "hog_665"
## [125] "min_of_day"               "cnn_39"
## [127] "hog_279"                  "hog_523"
## [129] "cnn_20"                   "hog_94"
## [131] "sd_pixel_val"             "hog_476"
## [133] "hog_797"                  "hog_342"
## [135] "hog_649"                  "hog_657"
## [137] "max_red"                  "doc2vec_2"
## [139] "hog_60"                   "edge_avg_value"
## [141] "punc_num_..2"             "hog_62"
## [143] "hog_819"                  "cnn_9"
## [145] "num_digit_chars"          "punc_num_..15"
## [147] "doc2vec_14"               "punc_num_..16"
## [149] "punc_num_."               "punc_num_..20"
## [151] "doc2vec_6"                "doc2vec_19"
## [153] "doc2vec_3"                "punc_num_..5"
## [155] "punc_num_..6"             "punc_num_..12"
## [157] "punc_num_..18"            "cnn_65"
## [159] "punc_num_..4"             "doc2vec_16"
## [161] "doc2vec_8"                "punc_num_..21"
## [163] "cnn_36"                   "punc_num_..11"
## [165] "punc_num_..26"            "punc_num_..13"
## [167] "punc_num_..27"            "doc2vec_7"
## [169] "cnn_0"                    "day"
## [171] "doc2vec_12"
```

**Take only the predictors we selected above**

```
reduced_new_training <- new_training[, c(final_predictors, 'growth_2_6')]
```

**Training our model**

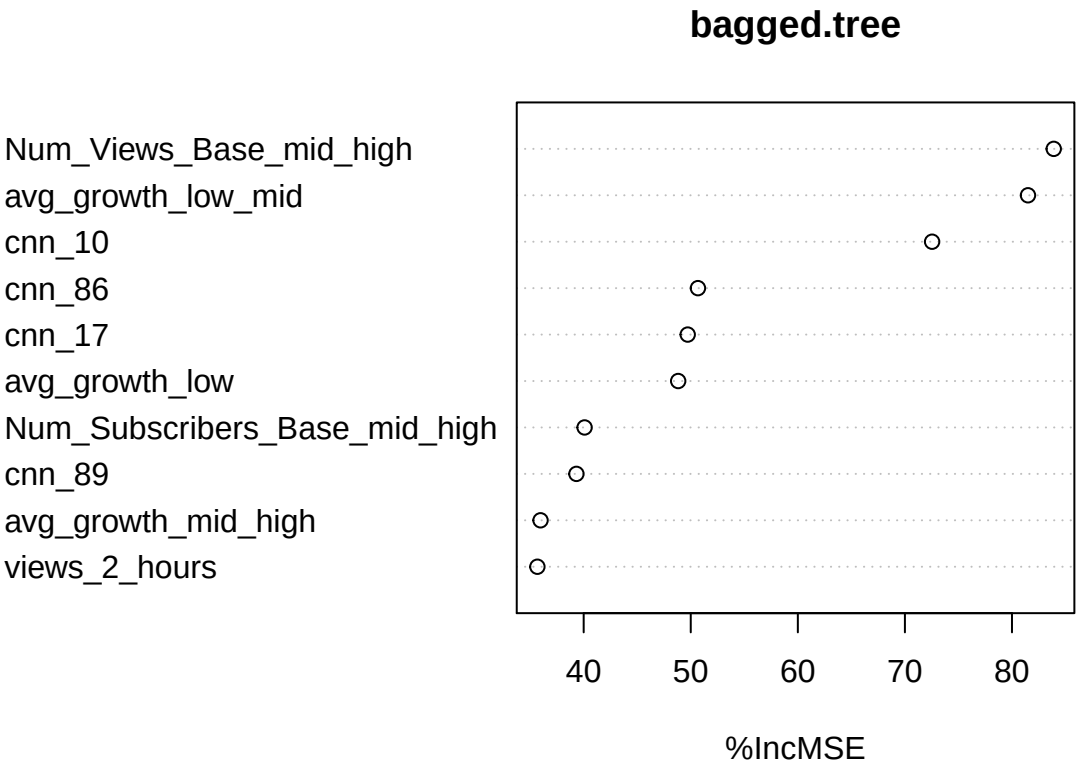
```
set.seed(999) # change seed to 12 for random forest
bagged.tree <- randomForest(growth_2_6 ~ .,
                            data = reduced_new_training,
                            mtry = dim(reduced_new_training)[2] - 1, # change to 165 for random forest
                            ntree = 500,
                            importance = T)
print(bagged.tree)
```

```
##
## Call:
##  randomForest(formula = growth_2_6 ~ ., data = reduced_new_training,      mtry = dim(reduced_new_tra
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 171
##
##           Mean of squared residuals: 2.212858
##                     % Var explained: 68.02
```

```
bagged.tree.predictions <- predict(bagged.tree, reduced_new_training)
bagged.tree.rmse <- sqrt(mean((bagged.tree.predictions - reduced_new_training$growth_2_6)^2))
bagged.tree.rmse
```

```
## [1] 0.5773139
```

```
varImpPlot(bagged.tree, n.var = 10, type = 1)
```




**Making predictions on test data**

```
# create dataframe with two columns, id and predictions
submit <- data.frame(read.csv("stats101c-lec4-final-competition/test.csv")$id,
                     predict(bagged.tree, new_testing))

# name columns to comply with rules
colnames(submit) <- c("id", "growth_2_6")

# write to csv file, to be submitted to Kaggle
# write.csv(submit, "Final_submit_this.csv", row.names=FALSE)
```

**Code to create Figure 1**

```
library(ggplot2)

training <- read.csv("stats101c-lec4-final-competition/training.csv")

plot1 <- ggplot(training, aes(x = Duration, y = growth_2_6)) +
   geom_point() +
   labs(caption = "Figure 1: Duration vs growth_2_6 Plot") +
   theme(plot.caption = element_text(hjust = .5))

plot1
```

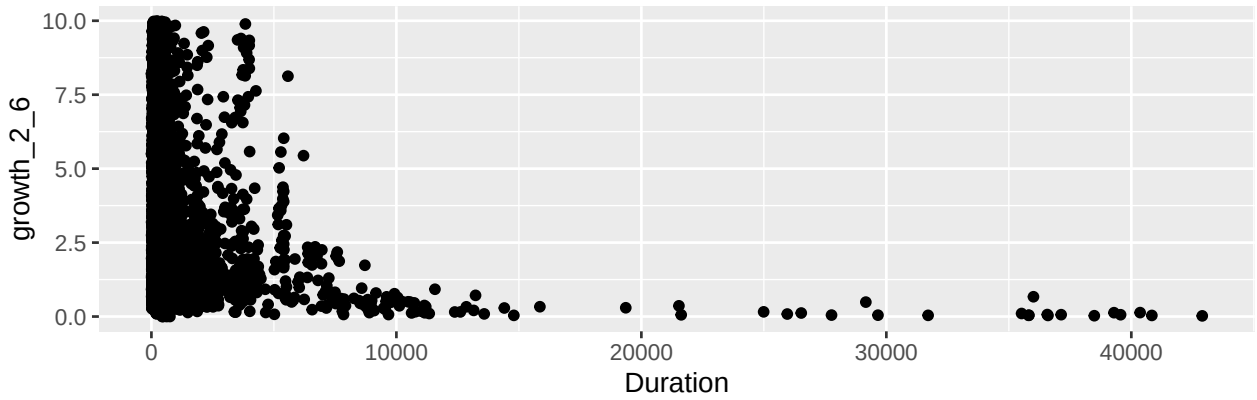



Figure 1: Duration vs growth_2_6 Plot

## Statement of Contribution

Darren: I performed the EDA and shared the results with Yonatan. I found that bagging on the uncleaned, original dataset led to pretty good results. After Yonatan cleaned the data, I tried out various thresholds for our find_pred() and tried out the resulting predictors in bagging.

Yonatan: I also performed EDA and shared the results with Darren. I cleaned the dataset (removing *id*, using *PublishDate* to create new variables, etc). I also tried using different values of $m$ for our model.