Predicting Growth Rate of Youtube Video Views

Qinyi Chen, Yue Wu, Hanyue Zhang

December 13th 2020

1 Introduction

Youtube is the most popular video-sharing platform in the world. More people become content creators on Youtube and make money based on the views of their videos. For the new content creators, it is necessary to know how to make the video grow faster. Since the video's early growth pattern can be one significant indicator to predict whether the videos can have eventual success or not, the views growing in the first few hours should be considered as valuable. Therefore, in order to know how it works, we are going to predict the percentage change in views on a video between the second and sixth hour after its publishing, so that the new content creators can improve the quality of their video production based on this metric.

2 Preprocessing

The training dataset collects all the information from four categories, which are Thumbnail Image Features, Video Title Features, Channel Features, and Other Features. This comprehensive dataset contains a total of 260 variables and 7242 observations. Before making any modifications to the dataset, we randomly split 30% of the data as our validation data to avoid bias when testing the model's performance, and we use the other 70% of the dataset as the new training data to process the data transformation and feature selection.

In order to improve the explanatory power of the corresponding features, we apply the data transformation for some variables. The character variable named "PublishedDate" indicates when the content creator publishes a video on YouTube, including the date and time. Intuitively, both date and time might provide significant effects on predicting views. Thus, we split the "PublishedDate" variable into two different variables, which are "month" and "hour". Since time is continuous, we consider converting them into numeric variables. The Method 1 below shows the specific details of how we transform the character information into the corresponding number. The "month" is a numeric variable that converts each date into a meaningful number. Based on Method 1: Formula 1 and Ex1, December 12^{th} would approximately become 12.39, by doing the addition of the number of the month and the percentage of how many days we passed off this month. The other variable "hour" containing the time would also be converted to numeric similarly according to Method 1: Formula 2. As an example, in Method 1: Ex 2, 3:20 can be approximated as 3.33.

Method 1

$$Formula1: "month" = Month + \frac{Date}{days\,of\,this\,month}$$

$$Ex1: December\ 12^{th} = 12 + \frac{12}{31} = 12.39$$

$$Formula2: "hour" = Hour + \frac{Date}{60}$$

$$Ex2: 3:20 = 3 + \frac{20}{60} = 3.33$$

Table 1 shows an example of dummy variables in the original data and the combined categorical variable

avg growth low	avg growth low mid	avg growth mid high	avg growth
1	0	0	1
0	1	0	2
0	0	1	3
0	0	0	4

Table 1: Example of Combine Levels

There are 12 dummy variables of Channel Features describing four features: number of subscribers, number of total views, average percent change in views on a video between the 2^{nd} and 6^{th} hour among all videos, and number of other videos in this dataset belonging to this channel. Although each feature contains four levels which are low, between low and medium, between medium and high, and high, there are only three variables in each feature to distinguish these levels. The three variables for each feature only represent the first three levels. For example, the first three columns in Table 1: Example of Combine Levels, contains three variables for the feature "avg_growth". The value of 1 indicates that the observation belongs to the corresponding level, while the value of 0 indicates that the observation does not belong to that level. If all three variables are 0, the observation belongs to the level of high according to data description. Since some variables might contain too many zeros and consider as meaningless due to the low variance, we merge them into one categorical variable to avoid this disadvantage. As an example, the fourth column in Table 1: Example of Combine Levels, shows the merged variable of previous three variables.

3 Feature Selection

3.1 Removing features with low-variance

If the variance of one variable is very low or close to zero, it indicates that this variable cannot provide significant contributions to the prediction of the model. Since it cannot improve the predicting performance, it should be removed. Here we set 100 non-zero observations of each column as our baseline approach to removing features with low-variance, then we remove all variables that contain no more than 100 non-zero observations. As a result, we delete 31 variables from the training dataset.

3.2 Feature selection based on Lasso

After removing the low-variance features, there are still 220 predictors in the training dataset. Since the number of variables is quite large, we use the Lasso to perform variable selection, overcoming the disadvantage of involving all the predictors in the model. The Lasso produces sparse models that involve only a subset of the variables, which helps us better assess which variables play a significant role. Since Lasso is one of the shrinkage methods that force some nonsignificant variables' coefficients towards 0, we remove all nonsignificant variables as our second baseline approach to further feature removal. Since Lasso separates the categorical variables into different levels and treats each of them individually, some levels are considered as significant, and others are not. As long as one level makes sense, we still decide to keep the original variable.

3.3 Feature Selection based on Random Forest

After performing feature selection using the Lasso method, the dataset contains 114 variables which are still considered as high dimensional. Thus we prefer to do a collation on the dataset using the Random Forest method to select the most high-predicting power variables. Since the dataset is high dimensional and it includes a large number of correlated predictors, the Random Forest method going through all possible splits on a smaller predictor subset size m is usually helpful. Therefore, we use a typical $m = p/3 = 37$ to construct a regression Random Forest model, and the variance explained by the model is 68.06%. The variable's importance of the Random Forest model gives us some potential insight into the variables predicting power. We include the top 25% of variables that have a high average decrease of accuracy in prediction when it is excluded from the model and the top 25% of variables that have a large increase in node impurity from splitting. Specifically, we have 28 variables that satisfy the first criteria and 28 variables that meet the second criteria. Lastly, we only keep 26 variables, that satisfy both conditions, to be the final predictors.

4 Statistical Model

In order to select the final model with the best predictive capability, we apply six methods: Multivariate Linear Regression, Lasso, Ridge, Bagging, Random Forest, Boosting, to the training dataset, and compute

the root mean squared error on both training dataset and validation dataset. For the Random Forest model specifically, since the optimal values for parameter m, the size of the smaller predictor subset, will vary in practice, we can treat m as a tuning parameter and use the out-of-bag error estimation, computationally less expensive approach, to select the optimal value m = 17.

Table 2 shows the comparison of the RMSE estimation from six methods.

Method	Validation	Training
Random Forest	**1.468522**	**0.592646**
Bagging	1.470501	0.587117
Lasso	1.678724	1.653955
Ridge	1.678905	1.654272
Linear Regression	1.679488	1.653883
Boosting	1.595208	1.453578

Table 2: RMSE Table

The Table 2: RMSE Table shows that the Bagging model has the lowest training RMSE, and the Random Forest model has the lowest validation RMSE. Since results based on validation data are more generalizing than training data, we consider that the Random Forest model with m = 17 gives the best prediction.

5 Result

As discussed in the Statistical Model section, according to Table 2: RMSE Table, the training RMSE of the final model is 0.592646 and the validation RMSE of the final model is 1.468522. Moreover, our final model's best evaluation metric value in the Kaggle public leaderboard is 1.39081.

6 Conclusion

In general, we think our model works well because of the effective data transformation and rigorous model choosing. Some variables might have the significant effect but in an unbefitting format on predicting the response variable, so we modestly convert the variables into the effective format to avoid the loss of partial or even entire information. The result of the subsequent feature selection confirms our assumption due to all of them being selected to perform the final model. Furthermore, during the process of choosing the final model, we compared 6 models, each of which is performing with the best parameter we statistically find. The final model we choose shows the most predictive capability according to the Table 2: RMSE Table, so we have no reason to doubt it is not the best model based on our predictors.

Nevertheless, there is still some place we can improve during the feature selection process. During the process of feature selection based on random forest, the variables for the final model are chosen based on the ranking of both standards in the importance table, yet we don't guarantee the number of variables we choose are the best for the final model.

7 Alternative Model

In the alternative model, we use the same procedure except the feature selection based on Random Forest. Instead, we process the feature selection based on Bagging. Unlike the Random Forest, only a subset of features is randomly selected, all the features are considered for splitting nodes in Bagging. In the final model, we keep using Random Forest with the out-of-bag error estimation, and it suggested the optimal value m = 14. The results show that the RMSE of the training dataset is 0.593485, and the RMSE of validation is 1.462921. Additionally, the metric evaluation value in the Kaggle public leaderboard is 1.39371. Since there might exist collinearity in the high-dimensional data, bagging performs not as well as random forest.

Contribution

Hanyue Zhang and Yue Wu performed data transformation on PublishedDate variable and data combination on 12 dummy variables of Channel Features. Then Yue Wu and Qinyi Chen selected features based on variable's variance, Lasso method, and Random Forest method. Qinyi Chen and Hanyue Zhang constructed six alternative models and selected the final model. Hanyue Zhang and Yue Wu integrated all programming code and attached it to the Appendix. Qinyi Chen, Yue Wu, and Hanyue Zhang completed the final report and presentation PowerPoint together.

Appendix: R code

```
################################  Libraries Used ################################
library(caret)
library(glmnet)
library(randomForest)
library(tidyverse)
library(gbm)

################################ Import Data ################################
# import the data and remove id column from training dataset
training <- read.csv("training.csv")[-1]
testing <- read.csv("test.csv")


############################## Basic Data Exploration ##############################
dim(training)
str(training)
#determine whether training dataset contains any missing (NA/NULL) values
all(!is.na(training))
all(!is.null(training))


################## Split Data into Training Data and Validation Data #################
set.seed(6930)
trainIndex <- createDataPartition(training$growth_2_6, p = 0.7, list = FALSE)
train_data <- training[trainIndex, ]
validation <- training[-trainIndex, ]


############################## Data Transform and Combine ##############################
days <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

#########################################################################################
### input: dataset
### output: modified dataset
### purpose: convert publishedDate into two numeric variables: dates and times
###          we convert the month and day into dates, hour and minute into times
#########################################################################################
convert_published_date <- function(dataset) {
  # combine month and day
  # extract the month and day from PublishedDate
  month <- as.numeric(unlist(strsplit(dataset$PublishedDate, "/"))
  [seq(1, nrow(dataset) * 3, by = 3)])
  date <- as.numeric(unlist(strsplit(dataset$PublishedDate, "/"))
  [seq(2, nrow(dataset) * 3, by = 3)])
  # combine the month and day into one numeric variable
  time <- month + date / days[month]
  # update the dataset
```

```
  dataset$month <- time

  # combine hour and minute
  # extract the time from PublishedDate
  time <- apply(matrix(dataset$PublishedDate), 1, function(x){
    unlist(str_split(x, " "))[2]
  })
  # combine the hour and minute into one numeric variable
  hour <- apply(matrix(time), 1, function(x){
    # extract the hour and minute
    time <- as.numeric(unlist(str_split(x, ":")))
    # combine the hour and minute
    time[1] + time[2]/60
  })
  # remove the PublishedDate variable
  dataset <- dataset[, -1]
  # update the dataset
  dataset <- data.frame(hour = hour, dataset)

  dataset
}

# convert publishedDate for training data
train_data <- convert_published_date(train_data)
# convert publishedDate for validation data
validation <- convert_published_date(validation)
# convert publishedDate for testing data
testing <- convert_published_date(testing)


################################################################################################
### input: dataset, channel features (string)
### output: modified dataset
### purpose: extract the variables describe the given feature
###          combine the variables into 1 variable that contains 4 levels
###          (low, low-mid, mid-high, high)
################################################################################################
combine_channel_feature_levels <- function(dataset, feature) {
  # determine which variables describe this feature
  index <- str_which(colnames(dataset), feature)
  # determine which level each observation belongs to
  determined_levels <- apply(dataset[, index], 1, function(x){
    # if any of the observation equal to 1, it belongs to the corresponding level
    # otherwise it belongs to high level
    ifelse(any(x == 1), which(x == 1), 4)
  })
  # modify the dataset, delete original channel feature columns
  dataset <- dataset[, -index]
  # update the dataset with the new variable
  dataset <- cbind(temp_name = as.factor(determined_levels), dataset)
  colnames(dataset)[1] <- feature

  dataset
}

# combine levels for Num_Subscribers
train_data <- combine_channel_feature_levels(train_data, "Num_Subscribers")
validation <- combine_channel_feature_levels(validation, "Num_Subscribers")
testing <- combine_channel_feature_levels(testing, "Num_Subscribers")
```

```r
# combine levels for Num_Views_Base
train_data <- combine_channel_feature_levels(train_data, "Num_Views_Base")
validation <- combine_channel_feature_levels(validation, "Num_Views_Base")
testing <- combine_channel_feature_levels(testing, "Num_Views_Base")
# combine levels for avg_growth
train_data <- combine_channel_feature_levels(train_data, "avg_growth")
validation <- combine_channel_feature_levels(validation, "avg_growth")
testing <- combine_channel_feature_levels(testing, "avg_growth")
# combine levels for count_vids
train_data <- combine_channel_feature_levels(train_data, "count_vids")
validation <- combine_channel_feature_levels(validation, "count_vids")
testing <- combine_channel_feature_levels(testing, "count_vids")


######################### Feature Selection Base on Variability ########################
# delete variables that have low variability
# have less than 100 non-zero values
remove <- names(which(colSums(train_data != 0) < 100))
train_data <- train_data[ , -which(colnames(train_data) %in% remove)]
validation <- validation[, -which(colnames(validation) %in% remove)]
testing <- testing[, -which(colnames(testing) %in% remove)]


######################### Feature Selection Based on lasso ##############################
set.seed(0617)
# The set up and define a grid of possible values for lambda
grid <- 10^seq(2, -3, by = -.1)
x <- model.matrix(growth_2_6 ~ ., train_data)[,-1]
y <- train_data$growth_2_6

# Select the best value for lambda using K-fold cross-validation.
cv.output <- cv.glmnet(x, y, family = "gaussian", alpha = 1,
                       lambda = grid, standardize = TRUE, nfolds = 10)
# Retrieve the actual best value of lambda.
best.lambda.cv <- cv.output$lambda.min
best.lambda.cv
# fit the best lasso model
lasso_reg <- glmnet(x, y, alpha = 1, lambda = best.lambda.cv, standardize = TRUE)

# delete the variables which suggested to be deleted by lasso
# delete variables
lasso_var <- c(rownames(lasso_reg$beta)[lasso_reg$beta[, 1] == 0])
train_data <- train_data[, -which(colnames(train_data) %in% lasso_var)]
validation <- validation[, -which(colnames(validation) %in% lasso_var)]


################### Feature Selection Based on Random Forest ##########################
set.seed(6930)
# fit a regression random forest model for feature section
rf.mod <- randomForest(growth_2_6~.,
                       data = train_data,
                       mtry = 37,
                       importance = TRUE,
                       ntree = 750)
# construct a tibble with variables names, %IncMSE, and IncNodePurity
t <- tibble(names = rownames(rf.mod$importance),
            mse = rf.mod$importance[, 1],
            node = rf.mod$importance[, 2])
# arrange the variables based on decreasing order of %IncMSE and IncNodePurity
```

```
arrange(t, desc(mse))
arrange(t, desc(node))

# get the variable names with top 25% based on the decresing order of %IncMSE
var1 <- arrange(t, desc(mse))$names[sort(t$mse, decreasing = TRUE) > quantile(t$mse, 0.75)]
# get the variable names with top 25% based on the decreasing order of IncNodePurity
var2 <- arrange(t, desc(node))$names[sort(t$node, decreasing = TRUE) > quantile(t$node, 0.75)]
# the final selected variables will the the variables both
# in the top 25% of %IncMSE and IncNodePurity
var <- c("growth_2_6", c(var1, var2)[duplicated(c(var1, var2))])

# variable importance plot of the top 25% variables
varImpPlot(rf.mod, n.var = 27)

# choose the variables for final model
train_data <- train_data[, which(colnames(train_data) %in% var)]
validation <- validation[, which(colnames(validation) %in% var)]

################################## Final Model ########################################
set.seed(18973846)
# Use the out-of-bag estimator to select the optimal parameter values.
# Here, we specify how we will evaluate our models
oob_train_control <- trainControl(method = "oob", savePredictions = TRUE)
tunegrid <- expand.grid(mtry = 1:26)
forestfit <- train(growth_2_6~.,
                   data = train_data,
                   method = "rf",
                   importance = FALSE,
                   trControl = oob_train_control,
                   tuneGrid = tunegrid)
# The model shows the best m
print(forestfit)

#############################################################################################
### input: true values, predicted values
### output: root-mean-square error
### purpose: compute root-mean-square error of predicted values
#############################################################################################
RMSE <- function(data, pred) {
  sqrt((1 / nrow(data)) * sum((data$growth_2_6 - pred)^2))
}

# check the RMSE for the result
forest.pred <- predict(forestfit, validation)
RMSE(validation, forest.pred)
forest.pred.train <- predict(forestfit, train_data)
RMSE(train_data, forest.pred.train)
rf.pred <- predict(forestfit, testing)


################################## Alternative Models ########################################
set.seed(0617)

# lasso
grid <- 10^seq(2, -3, by = -.1)
x <- model.matrix(growth_2_6 ~ ., train_data)[,-1]
y <- train_data$growth_2_6
tx <- model.matrix(growth_2_6 ~ ., validation)[,-1]
# cross validation determine lambda
```

```
cv.output <- cv.glmnet(x, y, family = "gaussian", alpha = 1,
                       lambda = grid, standardize = TRUE, nfolds = 10)
best.lambda.cv <- cv.output$lambda.min
best.lambda.cv
# best model
lasso_reg <- glmnet(x, y, alpha = 1, lambda = best.lambda.cv, standardize = TRUE)
# check the RMSE for the result
pred_lasso <- predict(lasso_reg, s = best.lambda.cv, newx = tx, type = "response")
pred_lasso_train <- predict(lasso_reg, s = best.lambda.cv, newx = x, type = "response")
RMSE(validation, pred_lasso)
RMSE(train_data, pred_lasso_train)

# ridge
set.seed(0904)
grid <- 10^seq(2, -3, by = -.1)
x <- model.matrix(growth_2_6 ~ ., train_data)[,-1]
y <- train_data$growth_2_6
tx <- model.matrix(growth_2_6 ~ ., validation)[,-1]
# cross validation determine lambda
cv.output <- cv.glmnet(x, y, family = "gaussian", alpha = 0,
                       lambda = grid, standardize = TRUE, nfolds = 10)
best.lambda.cv <- cv.output$lambda.min
best.lambda.cv
# best model
ridge_reg <- glmnet(x, y, alpha = 0, lambda = best.lambda.cv, standardize = TRUE)
# check the RMSE for the result
pred_ridge <- predict(ridge_reg, s = best.lambda.cv, newx = tx, type = "response")
pred_ridge_train <- predict(ridge_reg, s = best.lambda.cv, newx = x, type = "response")
RMSE(validation, pred_ridge)
RMSE(train_data, pred_ridge_train)

# boosting
boost.mod = gbm(growth_2_6 ~., data = train_data, n.trees = 750,
                    distribution = "gaussian")
# check the RMSE for the result
pred.boost = predict(boost.mod, validation, n.trees =750)
pred.mod.train = predict(boost.mod, train_data, n.trees =750)
RMSE(validation, pred.boost)
RMSE(train_data, pred.mod.train)

# MLR
lm.mod <- lm(growth_2_6 ~., train_data)
# check the RMSE for the result
lm.pred.train <- pred(lm.mod, train_data)
lm.pred <- pred(lm.mod, validation)
RMSE(validation, lm.pred)
RMSE(validation, lm.pred.train)
```