Prediction of Growth Rate of YouTube Videos

Chuwei Peng, Yujing Wen, Ziqian Liao

Introduction

As one of the largest video-sharing platforms nowadays, Youtube attracts a growing number of users to upload, view, and comment on videos. Knowing the growth pattern of a video within the first few hours has become an urgent need for content creators.

By incorporating methods of machine learning, we constructed a model which predicts the percentage change in views on a video between the second and sixth hour since its publishing. In this report, we presented methods we utilized to predict the response based on a dataset containing 7242 observations, each with measurements in 259 features that are contained in four categories: thumbnail image features, video title features, channel features, and other features.

Methodology

Preprocessing

We first examined the dataset for NA values, and found that this dataset is complete, so there was no treatment regarding NA. After that, we considered different aspects of feature engineering. More specifically, we included feature creation and feature selection methods, by doing which we excluded those trivial variables without missing essential ones.

i) Feature Creation

In our next step, we found that a column might be uninterpretable for our machine learning model -- the column *PublishedDate.* This column contained time tag information in which each unique value was treated as a new level of the factor (e.g. 4/17/2020 10:38). Instead of throwing away the column, we separated it into several new predictors (*month*, *day*, *year*, *hour*, and *min*) using the *separate* function in the r-library *tidyr* due to its potential significance in predicting the views. In the next step, we calculated the new variable *time* based on *hour* and *min* (the underlying relation was *time* = *hour**60 + *min*) using the *mutate* function to record the elapsed time in a day. The transformations results are shown in Figure 1.

	id <int>	PublishedDate <fctr>	year <int>	month <int>	day <int>	time <dbl>
1	0	4/17/2020 10:38	2020	4	17	638
2	1	8/31/2020 9:56	2020	8	31	596
3	2	8/16/2020 12:15	2020	8	16	735
			2020	8	22	540

Figure1. Transformation of Published Date Variable

There were two ways in which *month*, *day*, and *year* could be represented: they could be either factors or integers. Intuitively, these variables should be represented as factors since they could only switch between a limited amount of levels (e.g. 1 to 12 for *month*). However, such a simplification did cause a loss of useful information, as proved by results of the next stage of feature selection: when we treated them as factors, all three of them were completely filtered out. By contrast, when being treated as integers, some of them were kept in the next stage. This phenomenon is also intuitive based on common sense: Values of *month*, *day*, and *year* are not simply levels that are parallel with each other. There is an underlying order that contains important information. Therefore, we eventually decided to adopt these variables' integer forms.

ii) Feature Selection

In the third stage of the feature engineering, we filtered out variables without significant variance across observations. For example, *pct_nonzero_pixels*, *min_blue*, *min_red*, and *min_green* have only slight variance across different observations. We consider these variables useless in prediction. After several testing of RMSE with the following model, we set the threshold for variance as 0.015, and only kept variables with variance larger than this threshold.

Finally, we detected correlation/collinearity between the remaining variables by applying the correlation matrix and VIF. VIF produces a list of scores which quantify the severity of multicollinearity. We kept the names of the variables with large VIF for future reference. We also applied a correlation matrix and detected several variables with severe correlation (larger than 0.7). For example, *number_words* and *num_chars* have a relatively high correlation. After several trials of testing on training RMSE and Kaggle, we decided to combine the method of correlation matrix and VIF. Eventually we only kept 79 variables. In addition, we discarded trashy variables such as "id".

An alternative method that we touched on but did not include in our final method was principal component analysis (PCA). We observed no significant improvement in final score after using dimension reduction based on PCA. Given that PCA relies on orthogonal principal components, the fact that our data does not follow a multivariate distribution might be the cause of its failure.

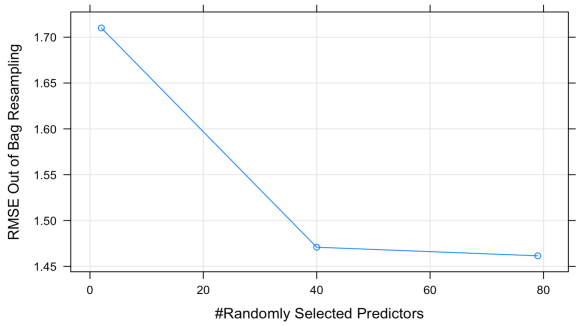
Statistical Model

i) Main Model Selection

In deciding on the model to use, we found that random forest performed well in dealing with data of high dimensionality. Its robustness to outliers and non-linear data well fits the dataset we are working with. However, one disadvantage of random forest is that it severely reduces the interpretability of the model. Given that we were driven to prefer accuracy over interpretability in this project, such a drawback was acceptable. In addition, random forest does a good job of assigning relative importance to the input features, so it also helps us achieve an insight to the story behind popular videos.

ii) Out-of-bag Error Estimate

After deciding on the main model, the remaining key problem was to decide on the value of *mtry*, the number of predictors selected in each node, so we examined the out-of-bag error estimates (OOB). Random forest uses subsampling with replacement to create training samples for the model to learn from. Consequently, when selecting a bootstrap sample, there always exist about 1/3 observations which are "out of the bag". Through calculating the error based on these observations, we conducted a free initial validation check that told us about the optimal values for the tuning parameter.




```
Random Forest

7242 samples
  79 predictor

No pre-processing
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared
   2    1.710126  0.5774002
  40    1.470860  0.6873806
  79    1.461497  0.6913482

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 79.
```

Figure2. OOB Cross-Validation Results of RMSE

As shown in the Figure 2, when *mtry* = 79, the OOB error achieved its minimum. In this case, the method utilized here was bagging since the model uses all the predictors.

Even though random forest outperforms bagging in most cases, here both our OOB error estimate results and Kaggle submission results indicated that bagging would be a better fit. We further observed that the Kaggle public score would normally be around 0.6 lower than the RMSE from OOB. This discovery confirmed that OOB RMSE is a relatively solid indicator of the model performance in prediction. As a result, we decided to use bagging instead of random forest as our final model.

Results

Our best evaluation metric value with this model as shown on Kaggle public leaderboard was 1.39949, which indicates that our model predicted well. We selected our second model, the one that was similar to our model#1 while the only difference being that we did not remove variables with severe correlation in the second. The second model got a score of 1.40228 on Kaggle public board.

Conclusions

In general, we believe that our carefully designed feature engineering methods make a significant contribution to the final success of our model. We acknowledged that including too many predictors in the model may lead to overfitting, while including too few may lead to information loss. Therefore, we tried to find a balance point for this bias-variance tradeoff through data tidying.

To find this balance, we tested on different thresholds in preprocessing and got various combinations of predictors. After several trials, we finally decided to use 79 (model#1: with variables of severe correlation removed) and 86 (model#2: with variables of severe correlation preserved) variables. We eventually got a private score 1.40654 for model#1 and 1.40287 for model#2. Trivial differences between the private and public scores (0.007 for model#1, 0.004 for model#2) for our two models indicated the stability of our method. It is reasonable to foresee that the robustness and stability of our model would lead to outstanding performance in predicting future datasets.

A way to further improve on our model is to train on parameters other than *mtry*. We only use the default number of 500 for *ntree*, that is, the number of trees to grow. Tuning this parameter may bring better results, but we did not train it since training on parameters generally costs lots of time and computational power, while we were already optimizing *mtry*, we could not afford training one more. For future improvements, we may graph different *ntree* values and their corresponding error to find the ideal *ntree*.

Also, by comparing two models, we found out that model#1 had a slightly lower private score than model#2, as being contradicted to our expectation. However, we still believe that removing variables of severe collinearity can reduce the noise in data and boost our accuracy, and that model#1 still has the potential to remove variables which bring noise: when two variables are severely correlated, we have to decide which variable to keep. This decision may be essential since their predictive power might be different. Given that the RMSE score depends largely on the random set of testing data, perhaps we will find that model#1 performs well in future trails.

Team Contribution

Chuwei Peng: Data Cleaning, Feature Selection, Report Writing, Presentation Creating
Yujing Wen: Feature Selection, Model Building, Report Writing, Presentation Creating
Ziqian Liao: Feature Creation, Model Building, Report Writing, Presentation Creating

Appendix

FEATURE ENGINEERING

```{r }
set.seed(1)
library(leaps)
train<-read.csv("training.csv")
library(tidyr)
library(dplyr)
sum(is.na(train))
#tidying, transforming published date variable
train.tidy <- train %>% separate(PublishedDate, into = c("date", "time"), sep = " ") %>%
separate(date, into = c("month", "day","year"), sep = "/") %>% separate(time, into = c("min",
"sec"), sep = ":")
train.tidy$day <- as.integer(train.tidy$day)
train.tidy$month <- as.integer(train.tidy$month)
train.tidy$year <- as.integer(train.tidy$year)
train.tidy$hour <- as.integer(train.tidy$hour)
train.tidy$min <- as.integer(train.tidy$min)
train.tidy <- train.tidy %>% mutate(time = hour*60 + min)
train.tidy <- train.tidy %>% select(-c(id,hour,min))
train=train.tidy
#remove small variance
variances <- diag(var(train))
train=train[,which(variances > 0.015)]
#VIF
formula <- as.formula(growth_2_6~.)
fit <-lm(formula, data = train)
#the linearly dependent variables
ld.vars <- attributes(alias(fit)$Complete)$dimnames[[1]]
#remove the linearly dependent variables variables to check vif
formula.new <- as.formula(
   paste(
      paste(deparse(formula), collapse=""),
      paste(ld.vars, collapse="-"),
      sep="-"))
#run model again
fit.new <-lm(formula.new, data = train)
library(plyr)
library(car)
# Set a VIF threshold. All the variables having higher VIF than threshold
#are dropped from the model
threshold=2.5
# Sequentially drop the variable with the largest VIF until all variables have VIF less than
threshold
drop=TRUE
aftervif=data.frame()
```

```
while(drop==TRUE) {
  vfit=vif(fit.new)
  aftervif=rbind.fill(aftervif,as.data.frame(t(vfit)))
  if(max(vfit)>threshold) {
  fit.new=update(fit.new,as.formula(paste(".","~",".","-",names(which.max(vfit))))) }
  else {
    drop=FALSE
  }
}

print(fit.new)
vfit_d= as.data.frame(vfit)
setdiff(colnames(train),rownames(vfit_d))

#detect collinearity with correlation matrix
corr<-cor(train)
for(i in 1:ncol(train)){
current<-cor(train)[,i]
current<-current[-i]
if(length((names(current)[which(abs(current)>0.7)])!=0)){
cat(colnames(train)[i],":",(names(current)[which(current>0.7)]),"\n")
  }
}
#After several trails, deciding to kick out these variables
#FOR THE MODEL WITH BETTER RMSE SCORE ON PRIVATE BOARD, BUT WORSE
SCORE ON PUBLIC BOARD, THIS LINE OF REMOVING SEVERELY COLLINEAR
VARIABLES IS REMOVED BUT EVERYTHING ELSE IS THE SAME
notkeep_names<-c("hog_13","hog_139","cnn_17","sd_red","sd_pixel_val",
"punc_num_..8","num_words")

#remove severely collinear variables
col_train<-colnames(train)
 index_notkeep<-c()
for (i in 1:length(col_train)){
  for(j in 1:length(notkeep_names)){
   if (col_train[i]==notkeep_names[j]){
    index_notkeep<-c(index_notkeep,i)
  }}
}
 train=train[,- index_notkeep]
```

Out Of Bag

```{r}
library(caret)
myTr <- trainControl(method = "oob")
model.cv.2 <- train(growth_2_6~., method = 'rf', data =train,trControl = myTr,importance=TRUE)
```

```
plot(model.cv.2)
model.cv.2
```

BAGGING

```{r}
#read test data
test<-read.csv("test.csv")
test.tidy <- test %>% separate(PublishedDate, into = c("date", "time"), sep = " ") %>%
separate(date, into = c("month", "day","year"), sep = "/") %>% separate(time, into = c("min",
"sec"), sep = ":")

test.tidy$day <- as.integer(test.tidy$day)
test.tidy$month <- as.integer(test.tidy$month)
test.tidy$year <- as.integer(test.tidy$year)
test.tidy$min <- as.integer(test.tidy$min)
test.tidy$sec <- as.integer(test.tidy$sec)

test.tidy<- test.tidy %>% mutate(time = min*60 + sec)
test.tidy<- test.tidy %>% select(-c(id,min,sec))
test=test.tidy
library(randomForest)
set.seed(1)
rf.79 <- randomForest(growth_2_6 ~ ., data = train,mtry = 79, importance=TRUE)
#FOR THE MODEL WITH BETTER RMSE SCORE ON PRIVATE BOARD, BUT WORSE
SCORE ON PUBLIC BOARD, mtry is 86 BECAUSE  EVERYTHING ELSE IS THE SAME
EXCEPT THAT WE DID NOT REMOVE MULTICOLLINEARITY
mypred <- predict(rf.79, newdata = test)
sample<-read.csv("sample.csv")
sample$growth_2_6 <- mypred
result.12.04_5 <- sample
write.csv(result.12.04_5,"womendeguang3.csv", row.names = F)
varImpPlot(rf.79)
```

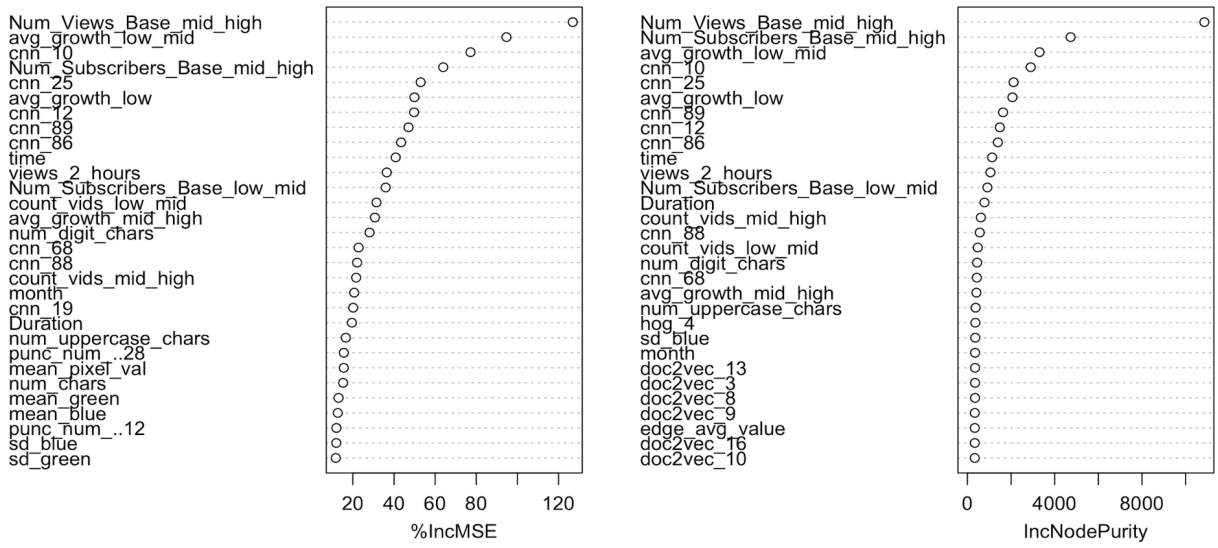


Figure3. Variable Importance Plot of Random Forest Model for Presentation