Prediction of YouTube Video Growth Rate

Stats 101C Final Project

Atif Farook, Julia Wood, Conner Ching

Contents

I. Introduction

The success of content creators on YouTube is largely driven by the number of views their videos receive; views are a direct measure of engagement and help determine a video's earning potential. Consequently, content creators try to forecast the virality of their videos as it might indicate eventual success and overall channel health. In this project, we aim to predict virality by using the percentage change in views between the second and sixth hour since a video's publishing. We use several video features such as thumbnail image, video title, channel parameters, duration, and more.

II. Methodology

a. Preprocessing

i. Exploratory Data Analysis (EDA)

We started by conducting Exploratory Data Analysis (EDA) to discover patterns and spot anomalies with the help of summary statistics. We noticed that some **predictor columns contained only zeroes and had negligible values of variance** [32 predictors in total]. After removing them from the dataset, we focused on **removing correlated variables** [correlation > 0.9].

ii. Defining New Variables

Additionally, based on the first few models, we noticed that the boolean vectors denoting if a video had a low/low-mid/mid-high subscriber base, view base, average growth rate, and video count were significant predictors. Thus, for each of these different metrics, we created a single factor with multiple levels: 1 - denotes a 1 in the "low" column; 2 - denotes a 1 in the "low-mid" column; 3 - denotes a 1 in the "mid-high" column. Essentially, we created four new categorical predictors out of twelve boolean predictors - *Num_Subscribers, Num_Views, avg_growth, and count_vids*. This allowed us to **centralize and strengthen the predictive power of these variables**.

iii. Feature Selection

We then focused on narrowing down predictors. After using basic processing methods (transformation and combining predictors) and dimensionality reduction methods (based on correlation and variance) we were still left with 100+ predictors. It was **difficult to obtain great results by with unsupervised methods** like PCA as they **failed to take into account the information that existed between feature values and the target variable** (growth_2_6). Since we found initial success with random forests we searched for feature selection methods specific to them. We ultimately decided on using the Boruta Algorithm - a wrapper method for random forest models that captures the most important predictors in a dataset.

In this algorithm, the features of a dataset don't compete with each other, instead they compete with randomized versions of themselves. The algorithm creates identical columns but with shuffled values called "shadow features". It then records a threshold that is defined as the highest feature importance recorded among the shadow features. If one of the real features gets an importance level higher than this threshold, it is called a "hit" and each "hit" is recorded. The algorithm then uses this count of "hits" to create a binomial distribution [Appendix: Figure IV b.2.] representing an area of refusal (red bars in Fig 1.), irresolution (yellow bars in Fig 1.), or acceptance (green bars in Fig 1.) for each feature, allowing users to implement their own thresholds and determine how conservative they'd like their estimates to be.

Using the Boruta algorithm, we were able to narrow down to 47 predictors. As you can see in Fig 1., we were able to visualize the drop in importance for predictors and test them in groups (as shown on the graph). We had a total of 5 groups (no of predictors = 4, 8, 33, 47, and 95) and evaluated their respective RMSE before settling on the group with 47 predictors. [More information on the Boruta Algorithm can be found in Appendix b.]

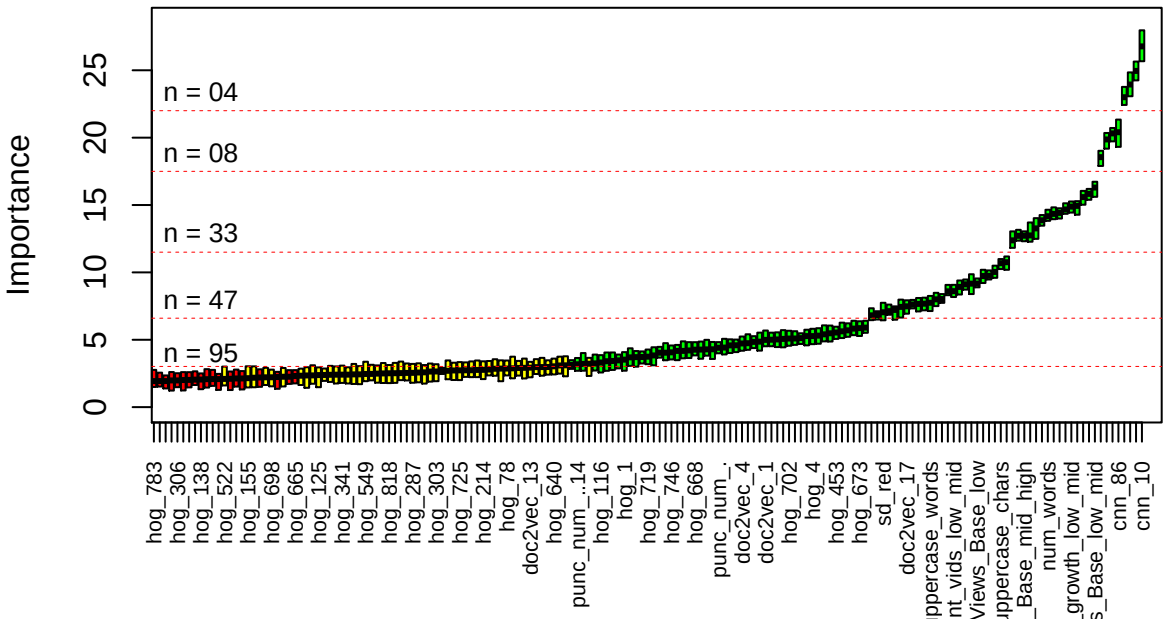

Figure 1: Importance values from boruta algorithm

b. Statistical Model

i. Tuning

As previously mentioned, after exploring various models (gradient boosting, linear regression, SVM), we found the most success with random forest models. Since the botura algorithm gave us the 47 best predictors we now needed to focus on tuning the model.

To determine the number of variables available for splitting at each tree node **(mtry)**, we used out-of-bag cross validation and chose the number of trees using RMSE. Next, using a loop we tried to find the best number of trees **(ntree)** to grow and the minimum number of observations in a terminal node **(nodesize)**.

ii. Model Description

After tuning our final model is based on 47 predictors (determined using the Boruta algorithm) with mtry = 15, ntree = 500, and nodesize = 5. Essentially, this means that to predict growth_2_6 we used an ensembling machine learning algorithm that creates multiple decision trees - using the parameters above - and then combines the output generated by each of the decision trees.

A decision tree is a classification or regression model which works on the concept of information gain at every node. For example, a single value can run through the entire tree based on some binary classification (T/F) until it reaches a terminal node; the final prediction is the average of the value of the dependent variable in that particular node. Figure 2 is an illustrative example of part of a decision tree that could be in our model. We see that at each node a decision is to be made and at the terminal nodes we have a prediction value.

The second model we submitted combined the above random forest model with a gradient boosting model (ntrees = 2516, interaction depth = 5, $\lambda = 0.01$). Gradient boosting is different from random forests in two ways: Random forests builds each tree independently while gradient boosting builds one tree at a time. This additive nature helps weak trees improve their shortcomings by introducing another weak tree. Secondly, random forests combine results at the end of processing by averaging while gradient boosting combines results along the way. **Since random forests achieve accuracy by reducing bias, and gradient boosting achieves accuracy by reducing variance, we hypothesized that averaging their predictions might help increase accuracy on both fronts**. Therefore, we predicted growth_2_6 with each model separately and then submitted the average of those predictions to Kaggle.

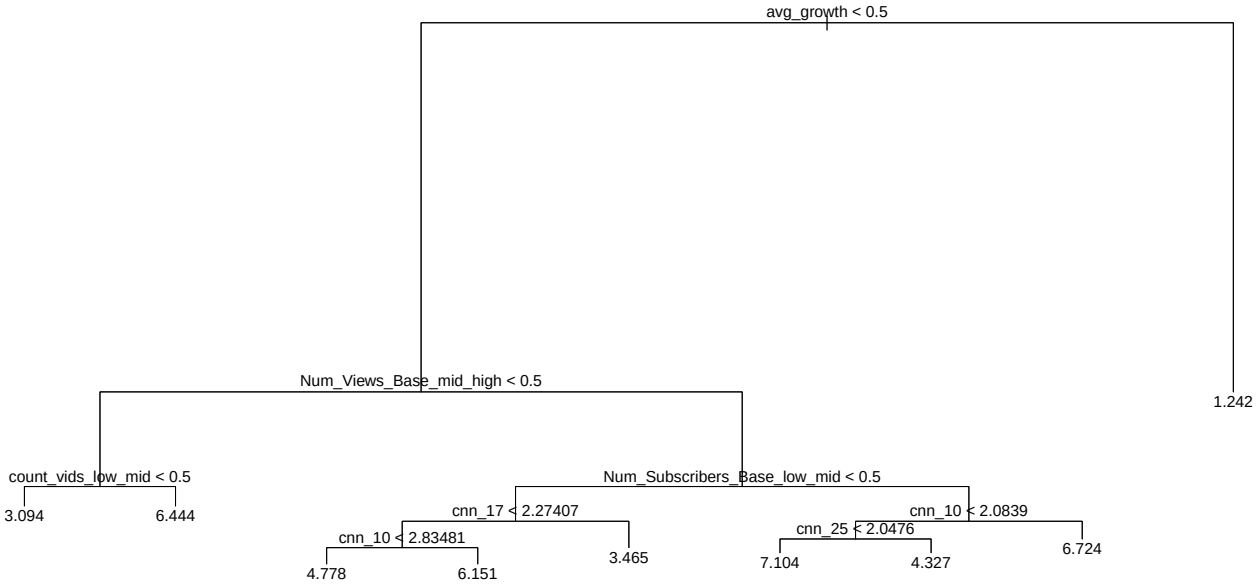

Figure 2: Example of a Decision Tree

III. Results

We evaluated our model using RMSE on validation data using a 70-30 split. Our final model resulted in an RMSE of 1.46. When applied to the testing data on Kaggle, the model resulted in an RMSE of 1.42. We opted for higher bias on our validation set as we wanted to avoid overfitting. Thus, we observed a lower RMSE on the testing data in Kaggle than our training RMSE in R. This reassured us that our model was not overfitted to the training data.

IV. Conclusions

Difference in scroes: An aspect we focused on while creating our model was overfitting. In the last kaggle competition, our model faired poorly on the private leaderboard vs the public one. To account for bias-variance tradeoff we used cross validation and tuning. Random forests are also innately less susceptible to overfitting. We can see that the steps we took worked as our private and public leaderboard scores only differed by 0.02.

Reasons for Success: Given that our model was not elaborate, we believe its success is due to the extensive preprocessing. In particular, we saw a significant decrease in the RMSE once we created the 4 categorical predictors (detailed in Methodology). Intuitively, these features — subscriber base, view base, average growth rate, and video count — are good proxies for a video's growth rate and should therefore be valuable predictors. Combining them allowed us to **centralize and strengthen the predictive power of these variables**.

In addition, utilizing the Boruta algorithm helped with feature selection as traditional random forest importance only measures of how much removing a variable decreases accuracy, and vice versa — by how much including a variable increases accuracy. This is not directly related to the statistical significance of the variable importance. By using the Boruta algorithm and its permuted copies, we are able to repeat random permutations and model both original and random attributes. This helps achieve more statistically significant results.

Areas of Improvement: Eager to optimize a model for submission, we didn't spend a significant amount of time exploring various models (SVM, linear regression, gradient boosting), given our early success with random forest. However, with further tuning, the other models could have led to more accurate predictions. Furthermore, we recognize that the creation of our new categorical variables could lead to multicollinearity.

V. Statement of Contributions

Atif Farook
Performed EDA. Helped create, train and tune the random forest model. Ran the boruta algorithm; split the precidors into various classes and tuned various random forest models. Created the graphs and wrote the Appendix section. Also, compiled the final document in Latex.

Julia Wood
I explored various feature selection mechanisms and built random forest models with various subsets of the predictors. I also had the idea of adding the four additional predictor variables for num_subscribers_high, num_base_high, etc., however these variables ended up being incorporated into our final models in a slightly different way (described in Methodology).

Conner Ching
I explored support vector machines and gradient boosting models through multiple Kaggle submissions before we ultimately decided to use the random forest model. I also wrote the majority of the report and partnered with Julia to create the slides for our presentation.

VI. Appendix

a. Code

```
## Load the libraries
library(tidyverse)
library(tidyr)
library(ggforce)
library(ggcorrplot)
library(class)
library(MASS)
library(reshape2)
library(grid)
library(gridExtra)
library(caret)
library(mclust)
library(boot)
library(MLeval)
library(ISLR)
library(glmnet)
library(gbm)
library(pls)
library(randomForest)
library(tree)
library(e1071)
library(Metrics)
library(VSURF)
library(Boruta)
library(ggraph)
library(igraph)
library(rpart.plot)
library(knitr)

## Loading and cleaning the data
## Load the dataset
data_main <- read.csv("training.csv")
```

```r
## Check if there are any NA's
any(is.na(data_main)  == TRUE)
data_nop <- data_main[,-c(1,2)]

## Check the percentage of 0's/NULL Values in each column
res <- colSums(data_nop==0)/nrow(data_nop)*100

## Names of every column
names <- names(data_nop)

## Variables to remove
var_remove <- c(c(names[which(res > 99)]), c("max_blue", "max_red", "max_green"))

## Variables to keep
vars_to_keep = names[!(names %in% var_remove)]
vars_to_keep <- c(vars_to_keep[-226], "growth_2_6")
data <- data_nop[,vars_to_keep]


# Removing varaiables that are highly correlated

df2 = cor(data_sd)
hc = findCorrelation(df2, cutoff=0.85) # putt any value as a "cutoff"
hc = sort(hc)
names(data_sd[,hc])

## Creating categorical variables for Num_Subscribers, Num_Views, avg_growth & count_vids

categories <- c("Num_Subscribers_Base_low", "Num_Subscribers_Base_low_mid",
                "Num_Views_Base_mid_high", "Num_Views_Base_low",
                "Num_Views_Base_low_mid", "Num_Views_Base_mid_high",
                "avg_growth_low", "avg_growth_low_mid", "avg_growth_mid_high",
                "count_vids_low", "count_vids_low_mid", "count_vids_mid_high")

Num_Subscribers <-  rep(0, nrow(data))
Num_Views <- rep(0, nrow(data))
avg_growth <- rep(0, nrow(data))
count_vids <-  rep(0, nrow(data))

##Num_Subscribers
i <- 1
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

Num_Subscribers[index1] <- 1
Num_Subscribers[index2] <- 2
Num_Subscribers[index3] <- 3
#table(Num_Subscribers)

## Num_Views
i <- 4
index1 <- which(data[,categories[i]] == 1)
```

```r
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

Num_Views[index1] <- 1
Num_Views[index2] <- 2
Num_Views[index3] <- 3
#table(Num_Views)

## avg_growth
i <- 7
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

avg_growth[index1] <- 1
avg_growth[index2] <- 2
avg_growth[index3] <- 3
#table(avg_growth)

##count_vids
i <- 10
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

count_vids[index1] <- 1
count_vids[index2] <- 2
count_vids[index3] <- 3
#table(count_vids)

add_categories <- cbind(Num_Subscribers, Num_Views, avg_growth, count_vids)
n <- ncol(data)

## Data set after preprocessing
data <- cbind(data[,1:(n-1)],add_categories, "growth_2_6" = data$growth_2_6)

## Boruta Algorithm

# Running the algorithm

# boruta <- Boruta(growth_2_6~., data = data, doTrace = 2)
# plot(boruta)
# save(boruta, file = "boruta_og.R")

# Load the file
load("boruta_og.R")

## Predictors that matter
getSelectedAttributes(boruta, withTentative = F)

## Selecting the predictors and ordering them
importance_df <- attStats(boruta)
ord <- order(importance_df$meanImp, decreasing = TRUE)
```

```r
importance_df[ord,]

pred <- rownames(importance_df[ord,])

# choosing the imp predictors after the second "drop" in graph: n = 47
n_pred <- 47
pred1 <- pred[1:n_pred]
pred1 <- c(pred1, "growth_2_6")

# Final data set after feature selection
data<- data[, pred1]

## Plot the graph
plot(boruta, xlab = "", xaxt = "n", outline=FALSE,
     whisklty = 0, staplelty = 0, ylim = c(0,28.5), xlim = c(70,229))
lz<-lapply(1:ncol(boruta$ImpHistory),function(i)
boruta$ImpHistory[is.finite(boruta$ImpHistory[,i]),i])
names(lz) <- colnames(boruta$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
at = 1:ncol(boruta$ImpHistory), cex.axis = 0.7)
abline(h = 22, col = "red", lty = 2, lwd = 0.5)
abline(h = 17.5, col = "red", lty = 2, lwd = 0.5 )
abline(h = 11.5, col = "red", lty = 2, lwd = 0.5 )
abline(h = 6.6, col = "red", lty = 2, lwd = 0.5 )
abline(h = 3.02, col = "red", lty = 2, lwd = 0.5 )

text(72, 4.02, "n = 95", cex = 0.8 )
text(72, 8, "n = 47", cex = 0.8 )
text(72, 12.9, "n = 33", cex = 0.8 )
text(72, 18.9, "n = 08", cex = 0.8 )
text(72, 23.4, "n = 04", cex = 0.8 )

#70% of data for train and 30% of data for test
train_size = floor(0.7 * nrow(data))

#set the seed
set.seed(1234)

#get training indices
train_ind = sample(seq_len(nrow(data)), size = train_size)

data_train = data[train_ind, ]
data_test = data[-train_ind, ]

X_train = model.matrix(growth_2_6~., data_train)[,-1]
y_train = data_train$growth_2_6

X_test = model.matrix(growth_2_6~., data_test)[,-1]
y_test = data_test$growth_2_6

mtry <- floor((ncol(data_train) - 1)/3)
```

```r
## Random Forest - Cross Validation + best values for mtry and ntree

## Might take a while to run ~ 5 minutes
# Fit classification tree using the 'randomForest' library.
set.seed(123)

# Use the out-of-bag estimator to select the optimal parameter values.
oob_train_control <- trainControl(method="oob",
                                  classProbs = TRUE,
                                  savePredictions = TRUE)

# We find the best value for m using cross validation
rf_default <- train(growth_2_6~. ,
                data = data_train, method = 'rf',
                trControl = oob_train_control)

rf_default

# Adjust Settings of Random Forest (If necessary)

## Might take a while to run  ~ 10 minutes+
# Grid search many different RF settings
mtry_vals = c(10, 16, 20, 25, 30, 47)
nodesize_vals = c(2, 5, 10, 20)

results = matrix(rep(0, length(mtry_vals)*length(nodesize_vals)),
                length(mtry_vals),length(nodesize_vals))
rownames(results) = apply(as.matrix(mtry_vals), 2,
                          function(t){return(paste("m =",t))})
colnames(results) = apply(as.matrix(nodesize_vals), 2,
                          function(t){return(paste("n =",t))})

for (i1 in 1:length(mtry_vals))
{
  m = mtry_vals[i1]
  for (i2 in 1:length(nodesize_vals))
  {
    n = nodesize_vals[i2]
    rf_model = randomForest(growth_2_6~., data = data_train,
                            ntree=400, mtry=m, nodesize=n, importance=TRUE)
    rf_preds = predict(rf_model, data_test)

    D = as.data.frame(cbind(data_test$growth_2_6, rf_preds))
    colnames(D) = c("true_growth_2_6", "rf_preds")

    test_set_r_sq = rmse(y_test, rf_preds)

    results[i1,i2] = test_set_r_sq

  }
}

recommended.mtry <- mtry ## uses (predictors/3)
```

```r
## can explore mtry = 24 as that was the best m from CV
tunegrid <- expand.grid(mtry=recommended.mtry)
set.seed(123)
oob_train_control <- trainControl(method="oob",
                                  classProbs = TRUE,
                                  savePredictions = TRUE)

forestfit.m <- train(growth_2_6~. ,
                     data = data_train, method = 'rf',
                     trControl = oob_train_control, tuneGrid = tunegrid,
                     ntree = 500)
print(forestfit.m, digits = 2)

#predict with RF after CV
rf_preds_cv = predict(forestfit.m, data_test)

test_set_r_sq = rmse(y_test, rf_preds_cv)

tree.diagram <- tree::tree(growth_2_6~. , data_train)
plot(tree.diagram)
text(tree.diagram)


test <- read.csv("test.csv")
sample <- read.csv("sample.csv")
data <- test

categories <- c("Num_Subscribers_Base_low", "Num_Subscribers_Base_low_mid",
                "Num_Views_Base_mid_high", "Num_Views_Base_low",
                "Num_Views_Base_low_mid", "Num_Views_Base_mid_high",
                "avg_growth_low", "avg_growth_low_mid", "avg_growth_mid_high",
                "count_vids_low", "count_vids_low_mid", "count_vids_mid_high")

Num_Subscribers <-  rep(0, nrow(data))
Num_Views <- rep(0, nrow(data))
avg_growth <- rep(0, nrow(data))
count_vids <-  rep(0, nrow(data))

##Num_Subscribers
i <- 1
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

Num_Subscribers[index1] <- 1
Num_Subscribers[index2] <- 2
Num_Subscribers[index3] <- 3
table(Num_Subscribers)

## Num_Views
i <- 4
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
```

```r
index3 <- which(data[,categories[i+2]] == 1)

Num_Views[index1] <- 1
Num_Views[index2] <- 2
Num_Views[index3] <- 3
table(Num_Views)

## avg_growth
i <- 7
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

avg_growth[index1] <- 1
avg_growth[index2] <- 2
avg_growth[index3] <- 3
table(avg_growth)

##count_vids
i <- 10
index1 <- which(data[,categories[i]] == 1)
index2 <- which(data[,categories[i+1]] == 1)
index3 <- which(data[,categories[i+2]] == 1)

count_vids[index1] <- 1
count_vids[index2] <- 2
count_vids[index3] <- 3
table(count_vids)
add_categories <- cbind(Num_Subscribers, Num_Views, avg_growth, count_vids)

test <- cbind(test, add_categories)
test <- test[,pred1[1:n_pred]]

## Used the CV Rf model
sample.prob <- predict(forestfit.m, test)

sample$growth_2_6 <- sample.prob

## FINAL DOCUMENT
write.csv(sample,"Final draft", row.names = FALSE)
head(sample)
```

b. Boruta Algorithm

Why the method or model is useful: The Boruta algorithm is a wrapper built around the random forest algorithm. It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable. It does this by implementing shadow features and iteration and decision criteria.

Shadow features: In this algorithm, the features of a dataset don't compete with each other, instead they compete with randomized versions of themselves. The algorithm creates identical columns but with shuffled values. It then records a threshold that is defined as the highest feature importance recorded among the shadow features. If one of the real features gets an importance level higher than this threshold, it is called a "hit" and each "hit" is recorded. This means that the importance of a feature is judged by whether it is capable of doing better than the best-randomized feature.

	age	height	weight	shadow_age	shadow_height	shadow_weight
0	25	182	75	51	176	75
1	32	176	71	32	182	71
2	47	174	78	47	168	78
3	51	168	72	25	181	72
4	62	181	86	62	174	86

Figure IV.b.1.: Example of how shadow features are created

Iteration and decision criteria: The algorithm does not use a hard threshold between refusal and acceptance. Instead, it implements binomial distribution based on the number of hits and divides it into three areas - an area of refusal, an area of irresolution, and an area of acceptance. This allows us to decide how conservative we want to be and also takes into account how the other variables in the model are performing.

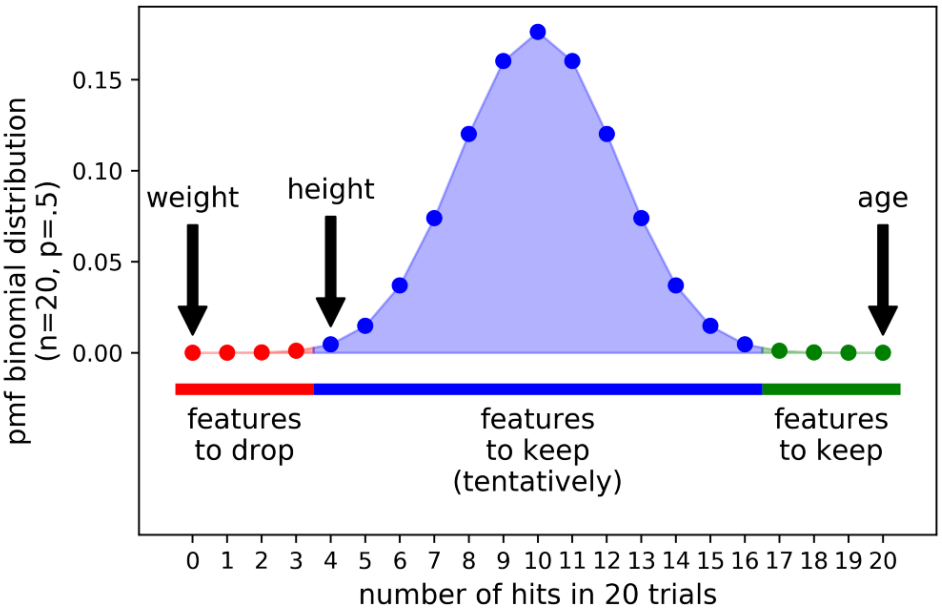

Figure IV.b.2.: Binomial distributuon of features based on "hits"

Advantages offered compared to methods seen in class: The Random forest importance is a measure of how much removing a variable decreases accuracy, and vice versa — by how much including a variable increases accuracy but in a way this is not directly related to the statistical significance of the variable importance. By

using the Boruta algorithm and its permuted copies, we are able to repeat random permutations and model both original and random attributes. This helps achieve more statistically significant results.

Additional Resources:

1. **Boruta Algorithm Explained**
   https://towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a
   Note: Images were referenced from this resource
2. **Boruta Algorithm In R** https://towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a
3. **CRAN Package - Boruta**
   https://cran.r-project.org/web/packages/Boruta/Boruta.pdf