Kaggle - Predicting Voting Outcomes

Greg Schreiter

July 14, 2016

Can we accurately predict voting outcomes by using informal polling questions?

Please note: this competition is only open to students of 15.071x - The Analytics Edge.

What predicts voting outcomes? In this competition, you'll be using data from Show of Hands, an informal polling platform for use on mobile devices and the web, to see what aspects and characteristics of people's lives predict how they will be voting for the presidential election.

Show of Hands has been downloaded over 300,000 times across Apple and Android app stores, and users have cast more than 75 million votes. In this problem, we'll use data from thousands of users and one hundred different questions to see which responses predict voting outcomes.

File descriptions

Here is a description of the files you have been provided for this competition:

- train2016.csv - the training set of data that you should use to build your models
- test2016.csv - the test set that you will be evaluated on. It contains all of the independent variables, but not the dependent variable.
- sampleSubmission2016.csv - a sample submission file in the correct format.
- Questions.pdf - the question test corresponding to each of the question codes, as well as the possible answers.

Data fields

- USER_ID - an anonymous id unique to a given user
- YOB - the year of birth of the user
- Gender - the gender of the user, either Male or Female
- Income - the household income of the user. Either not provided, or one of "under $25,000", "$25,001 - $50,000", "$50,000 - $74,999", "$75,000 - $100,000", "$100,001 - $150,000", or "over $150,000".
- HouseholdStatus - the household status of the user. Either not provided, or one of "Domestic Partners (no kids)", "Domestic Partners (with kids)", "Married (no kids)", "Married (with kids)", "Single (no kids)", or "Single (with kids)".
- EducationalLevel - the education level of the user. Either not provided, or one of "Current K-12", "High School Diploma", "Current Undergraduate", "Associate's Degree", "Bachelor's Degree", "Master's Degree", or "Doctoral Degree".
- Party - the political party for whom the user intends to vote for. Either "Democrat" or "Republican""
- Q124742, Q124122, . . . , Q96024 - 101 different questions that the users were asked on Show of Hands. If the user didn't answer the question, there is a blank. For information about the question text and possible answers, see the file Questions.pdf.

Load Packages

```
library(randomForest)
```

```
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.
```

```
library(rpart)
library(rpart.plot)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(e1071)
library(caTools)
library(flexclust)
```

```
## Loading required package: grid

## Loading required package: modeltools

## Loading required package: stats4
```

Introductory Analysis

We load the training and testing data:

```
setwd("C:/Users/Greg/Desktop/analytics edge")

train = read.csv("train2016.csv", na.strings = c("", "NA"), stringsAsFactors = F)
test = read.csv("test2016.csv", na.strings = c("", "NA"), stringsAsFactors = F)
```

First we perform a complete-case analysis:

```
CompleteCase=complete.cases(train)
```

We analyze how much data we have retained with this procedure:

```
sum(CompleteCase)/nrow(train)
```

```
## [1] 0.1251796
```

Since we only have 12.5 percent of the data as complete cases, we will lose too much information by simply deleting the observations with missing values. We will try several procedures for dealing with this: treating missing values as NA's, as well as filling in missing values with imputation or the mode of the variable. We will see what works best based on the results.

In order to use imputation, we must assume our data contains values that are "Missing at random" - when conditioned on all the data we have, any remaining missingness is completely random.

Data Cleaning

We first take some time to clean the data. This is critical for creating a model with good predictive power, as outliers and infeasible values will make the predictions inaccurate.

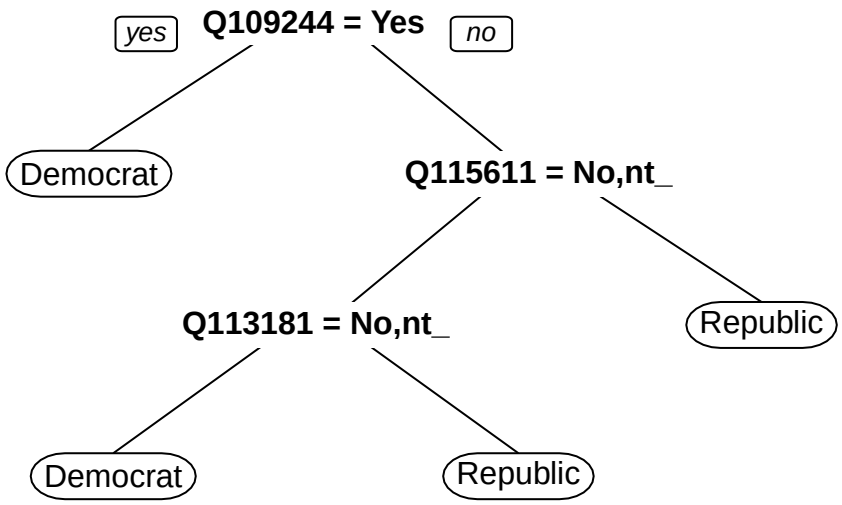
```
#Remove USER_ID, which has no predictive value, from training set
train$USER_ID <- NULL

# Replace NA's with median value
train$YOB[is.na(train$YOB)] <- 1983
test$YOB[is.na(test$YOB)] <- 1983
train$Gender[is.na(train$Gender)] <- as.character("Female")
test$Gender[is.na(test$Gender)] <- as.character("Female")
train$Income[is.na(train$Income)] <- as.character("$75,000 - $100,000")
test$Income[is.na(test$Income)] <- as.character("$75,000 - $100,000")
train$HouseholdStatus[is.na(train$HouseholdStatus)] <- as.character("Single (no kids)")
test$HouseholdStatus[is.na(test$HouseholdStatus)] <- as.character("Single (no kids)")
train$EducationLevel[is.na(train$EducationLevel)] <- as.character("Bachelor's Degree")
test$EducationLevel[is.na(test$EducationLevel)] <- as.character("Bachelor's Degree")

# Remove YOB outliers and infeasible values
train <- train[train$YOB >= 1935 & train$YOB <= 2000,]

# Set unanswered questions to have character value 'not_answered'
train[is.na(train)] <- as.character('not_answered')
test[is.na(test)] <- as.character('not_answered')

# Convert Party to a factor variable
```

```r
train$Party <- as.factor(train$Party)

# Write the train and test sets to a file for later use
write.csv(train, "imptrain.csv", row.names=FALSE)
write.csv(test, "imptest.csv", row.names=FALSE)

# Read in the data as factor variables
imptrain = read.csv("imptrain.csv")
imptest = read.csv("imptest.csv")
```

First Model: CART

We first start with a CART model. This model will show us which variables are most important:

```r
CART=rpart(Party ~ ., data=train, method="class" )

rpart.plot(CART)
```



We get 3 important variables from this selection process: Q109244 (Are you a feminist?), Q113181 (Do you meditate or pray on a regular basis?), and Q115611 (do you own a gun?). This makes sense, since these are some "hot-button" issues that help people decide which political party is right for them.

Basic Logistic Regression Models

```
logm1=glm(Party ~ Q109244 + Q113181 + Q115611, family="binomial", data = train)
logm1pred=predict(logm1, newdata=test, type="response")
logm1p=as.data.frame(cut(logm1pred, breaks=c(0,0.5,1), labels=c("Democrat", "Republican")))
lm1=data.frame(USER_ID = test$USER_ID, Predictions = logm1p)
colnames(lm1) <- c("USER_ID", "Predictions")
write.csv(lm1, "lm1.csv", row.names=FALSE)
```

This simple Logistic model gives us an accuracy of 0.62644 on the private leaderboard. Now we try adding one additional predictor, Income:

```
logm2=glm(Party ~ Q109244 + Q113181 + Q115611 + Income, family="binomial", data = train)
logm2pred=predict(logm2, newdata=test, type="response")
logm2p=as.data.frame(cut(logm2pred, breaks=c(0,0.5,1), labels=c("Democrat", "Republican")))
lm2=data.frame(USER_ID = test$USER_ID, Predictions = logm2p)
colnames(lm2) <- c("USER_ID", "Predictions")
write.csv(lm2, "lm2.csv", row.names=FALSE)
```

This yields a large increase in accuracy: 0.63937 on the private leaderboard. How does a model with all predictors perform?

```
logm3=glm(Party ~ ., family="binomial", data = train)
logm3pred=predict(logm3, newdata=test, type="response")
logm3p=as.data.frame(cut(logm3pred, breaks=c(0,0.5,1), labels=c("Democrat", "Republican")))
lm3=data.frame(USER_ID = test$USER_ID, Predictions = logm3p)
colnames(lm3) <- c("USER_ID", "Predictions")
write.csv(lm3, "lm3.csv", row.names=FALSE)
```

Not as well: Accuracy of only 0.61925.

Computer Selection of Important Variables

While manual selection of variables is a valuable tool, we may miss some important variables this way. We use the *caret* package to figure out which variables are the most important, based on repeated cross-validation:

```
control=trainControl(method="repeatedcv", number=5, repeats=3)
model=train(Party ~ . , data=imptrain, method="lvq", preProcess="scale", trControl=control)
```

```
## Loading required package: class
```

```
importance=varImp(model, scale=F)
print(importance)
```

```
## ROC curve variable importance
##
##   only 20 most important variables shown (out of 106)
##
##          Importance
## Q109244     0.6301
```

```
## Q115611     0.5864
## Q98197      0.5782
## Q113181     0.5743
## Gender      0.5538
## Q98869      0.5421
## Q99480      0.5366
## Q120472     0.5330
## Q106272     0.5315
## Q116881     0.5293
## Q110740     0.5274
## Q121699     0.5244
## Q120014     0.5212
## Q107869     0.5208
## Q108855     0.5198
## Q118232     0.5195
## Q101596     0.5170
## Q96024      0.5162
## Q120650     0.5156
## Q112270     0.5146
```

Now we use a similar procedure to select the model for us. We can also specify the type of model we want to use, so we will choose random forests:

```
# Using caret package to select predictors for random forest model using cross validation
control=rfeControl(functions=rfFuncs, method="cv", number=10)
results=rfe(imptrain[,c(1:5, 7:107)], imptrain$Party, rfeControl=control)
```

```
##
## Attaching package: 'plyr'

## The following object is masked from 'package:modeltools':
##
##     empty
```

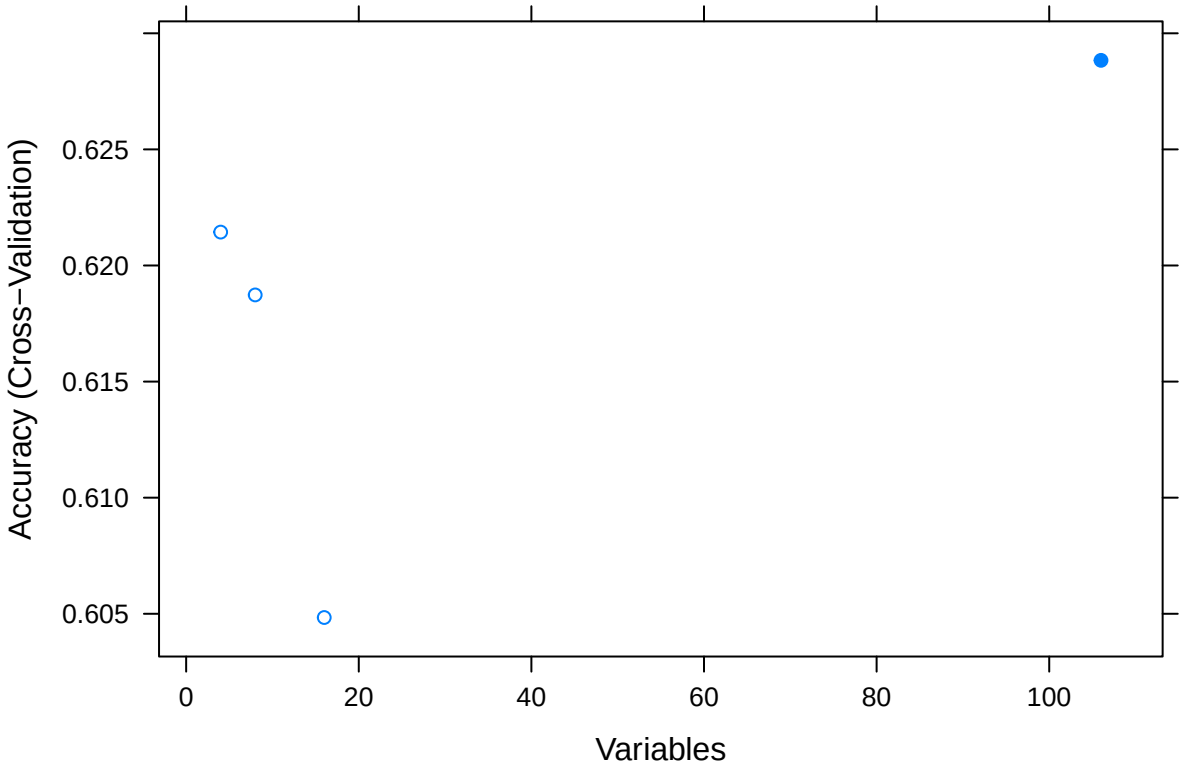
```
print(results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          4   0.6214 0.2303    0.01994 0.04006
##          8   0.6187 0.2302    0.01831 0.03642
##         16   0.6048 0.2037    0.01085 0.02143
##        106   0.6288 0.2504    0.01561 0.03092        *
##
## The top 5 variables (out of 106):
##    Q109244, Q115611, Q98197, Q113181, Gender
```

```
predictors(results)
```

```
##   [1] "Q109244"         "Q115611"         "Q98197"
##   [4] "Q113181"         "Gender"          "Q98869"
##   [7] "HouseholdStatus" "Q101163"         "Q106272"
##  [10] "Q120379"         "Q108855"         "Q120472"
##  [13] "Q106388"         "Q118233"         "Q121699"
##  [16] "Q119851"         "Q120650"         "Q115390"
##  [19] "Q105840"         "Q106042"         "Q99480"
##  [22] "Q106997"         "Q101596"         "Q106389"
##  [25] "Q123621"         "Q102089"         "Q108856"
##  [28] "Q114961"         "Q115899"         "Q118892"
##  [31] "Q120978"         "Q101162"         "Q110740"
##  [34] "Q96024"          "Q102674"         "Q123464"
##  [37] "Q98078"          "Q112270"         "Q107491"
##  [40] "Q116881"         "Q105655"         "Q122771"
##  [43] "Q104996"         "Q112478"         "Q121700"
##  [46] "Q116601"         "Q100010"         "Q121011"
##  [49] "Q120194"         "Q99716"          "Q116441"
##  [52] "Q114386"         "Q98059"          "Q100562"
##  [55] "Q122120"         "Q113992"         "Q118117"
##  [58] "Q117193"         "Q115195"         "Q108617"
##  [61] "Q114748"         "Q120014"         "Q116197"
##  [64] "Q120012"         "Q113583"         "Q98578"
##  [67] "Q108343"         "Q111848"         "Q112512"
##  [70] "Q115610"         "Q116448"         "Q116953"
##  [73] "Q99581"          "Q106993"         "Q107869"
##  [76] "Q108754"         "Q108342"         "Q109367"
##  [79] "Q122770"         "Q102687"         "Q117186"
##  [82] "Q119650"         "Q124122"         "Q116797"
##  [85] "Q99982"          "Q102906"         "Q124742"
##  [88] "Q100689"         "Q113584"         "Q122769"
##  [91] "Q102289"         "Q118237"         "Q100680"
##  [94] "Q115777"         "Q114517"         "EducationLevel"
##  [97] "Q115602"         "Q111220"         "Q119334"
## [100] "Q118232"         "Q114152"         "Q111580"
## [103] "Q103293"         "Q108950"         "YOB"
## [106] "Income"
```

```
plot(results)
```



This process tells us that we get the highest accuracy by using 8 variables: Q109244, Q115611, Q113181, Q98197, Gender, HouseholdStatus, Q98869, and Q121699. We will now use this as the starting point for our Random Forest models.

Random Forest Models

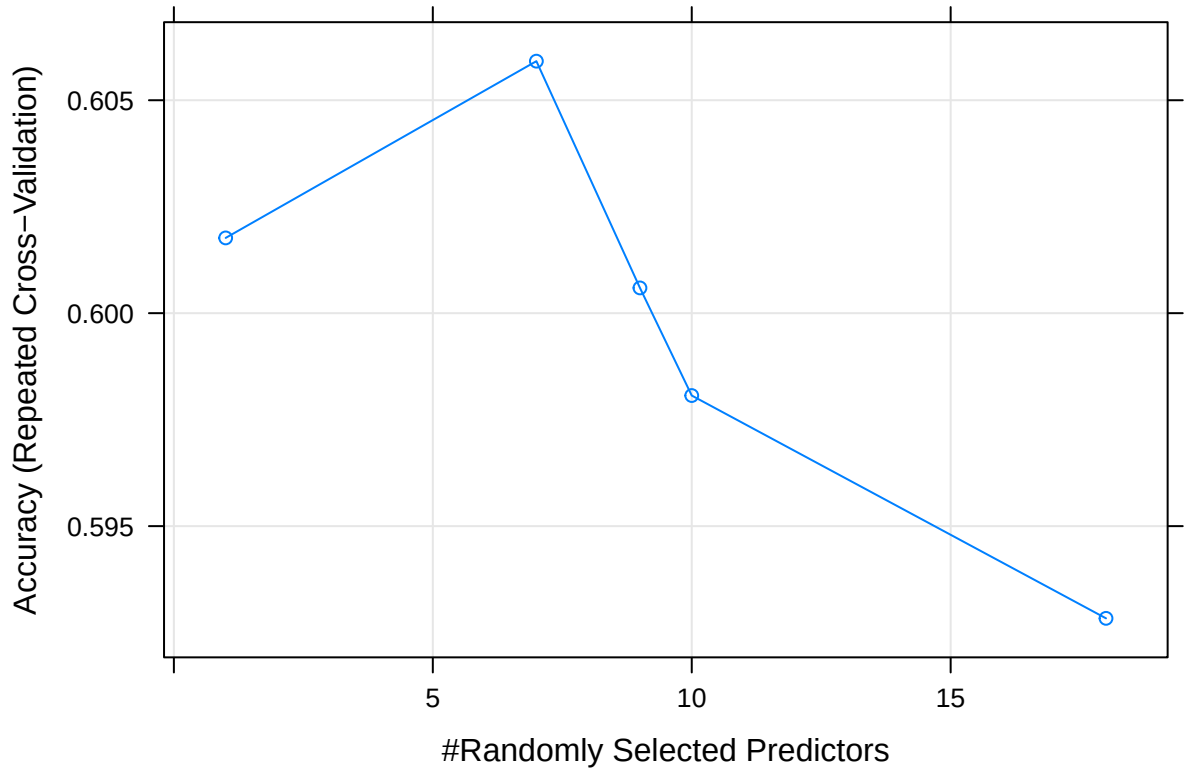
Using the information that the highest accuracy RF model was obtained by using 8 predictors, we will now attempt to tune both the *mtry* and *ntree* parameters.

```
# Tune mtry parameter
control <- trainControl(method="repeatedcv", number=5, repeats=2, search="random")
set.seed(1999)
mtry <- sqrt(ncol(imptrain))
rf_random <- train(Party~ Q109244+Q115611+Q113181+Q98197+Gender+HouseholdStatus+Q98869+Q121699, data=imp
print(rf_random)
```

```
## Random Forest
##
## 5542 samples
##  106 predictor
##    2 classes: 'Democrat', 'Republican'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 2 times)
## Summary of sample sizes: 4433, 4433, 4434, 4434, 4434, 4433, ...
```

```
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.6017671  0.1746221
##    7    0.6059170  0.2064279
##    9    0.6005932  0.1965830
##   10    0.5980659  0.1917972
##   18    0.5928346  0.1819528
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 7.
```

```
plot(rf_random)
```



The optimal value for mtry was found to be 7. Next, we use a for loop to iterate over the values of ntree:

```
# Split Training set
set.seed(123)
xt=sample.split(imptrain$Party, SplitRatio=0.7 )
train1=subset(imptrain, xt==TRUE)
train2=subset(imptrain, xt==FALSE)

# Tune ntree parameter
Acc=matrix(ncol=1, nrow=4000)
for (i in seq(1,4000,by=500)){
  rf=randomForest(Party ~ Q109244+Q115611+Q113181+Q98197+Gender+HouseholdStatus+Q98869+Q121699, data =
```

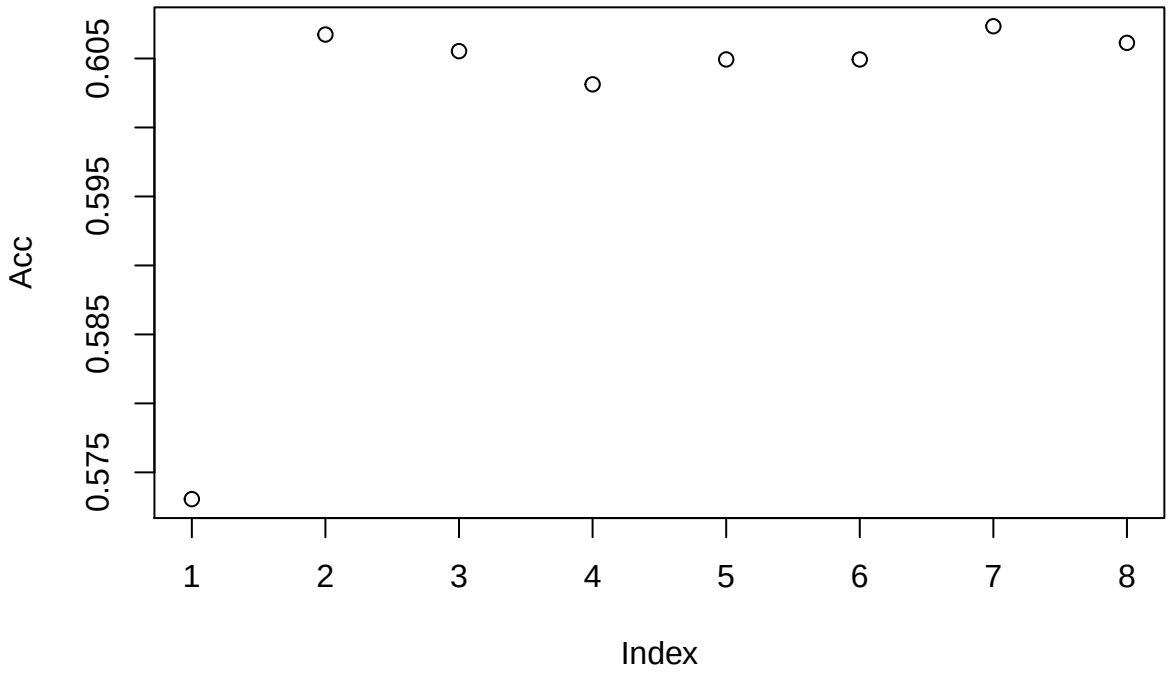
```
  pred=predict(rf, newdata=train2)
  tab=table(train2$Party, pred)
  Acc[i]=sum(diag(tab))/sum(tab)

}
Acc <- Acc[!is.na(Acc)]
max(Acc)
```

```
## [1] 0.6073361
```

```
which.max(Acc)
```

```
## [1] 7
```

```
plot(Acc)
```



The value of ntree that produces the maximum accuracy is $1 + 500 \times 7 = 3501$.

Using these two parameter values, we construct the Random Forest model using the 8 designated predictors:

```
rf1=randomForest(Party ~ Q109244+Q115611+Q113181+Q98197+Gender+HouseholdStatus+Q98869+Q121699, data = im

rfpred1=predict(rf1, newdata=imptest)
```

```
rfm1 = data.frame(USER_ID = test$USER_ID, Predictions = rfpred1)

write.csv(rfm1, "rfm1.csv", row.names=FALSE)
```

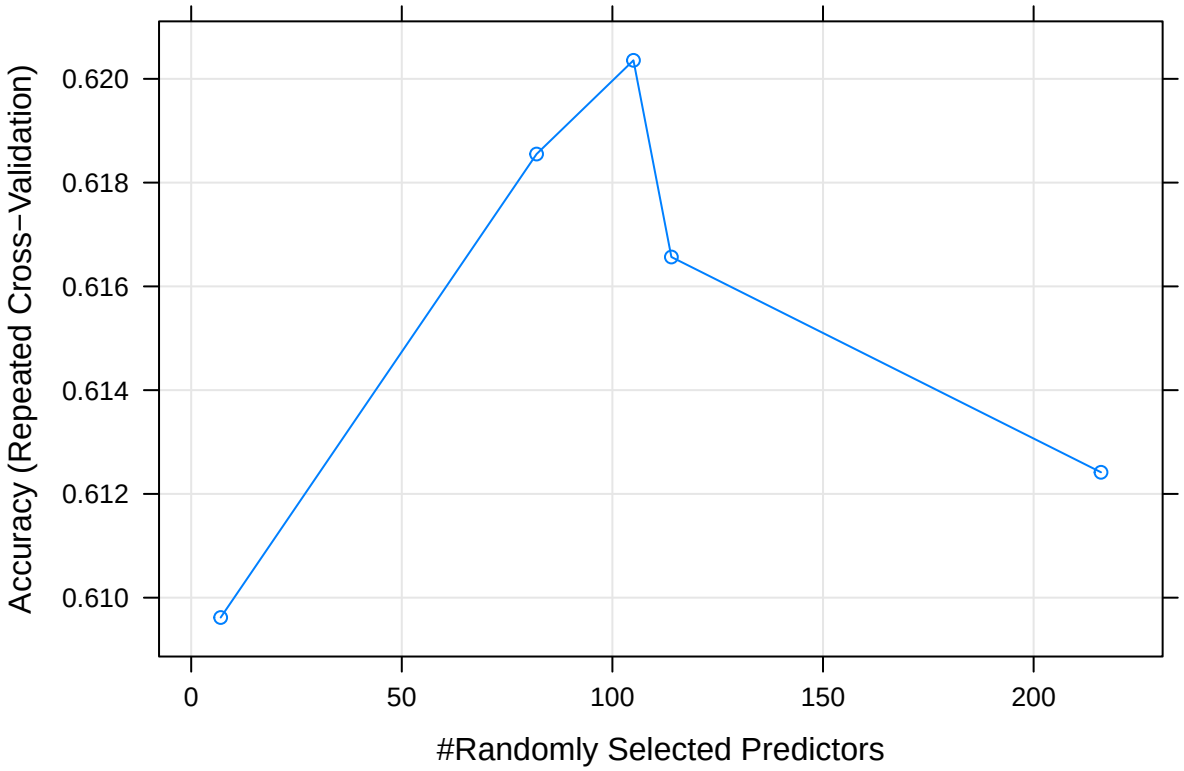
However, this model only scores 0.59483 on the private leaderboard. We have to do better than that!

Now we try a RF model with all predictors:

```
# Tune mtry parameter
control <- trainControl(method="repeatedcv", number=5, repeats=2, search="random")
set.seed(1999)
mtry <- sqrt(ncol(imptrain))
rf_random <- train(Party~., data=imptrain, method="rf", tuneLength=5, trControl=control)
print(rf_random)
```

```
## Random Forest
##
## 5542 samples
##  106 predictor
##    2 classes: 'Democrat', 'Republican'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 2 times)
## Summary of sample sizes: 4433, 4433, 4434, 4434, 4434, 4433, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     7   0.6096177  0.2061819
##    82   0.6185505  0.2330148
##   105   0.6203567  0.2369903
##   114   0.6165661  0.2294376
##   216   0.6124168  0.2204615
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 105.
```

```
plot(rf_random)
```



The optimal value for mtry was found to be 105. Next, we use a for loop to iterate over the values of ntree:

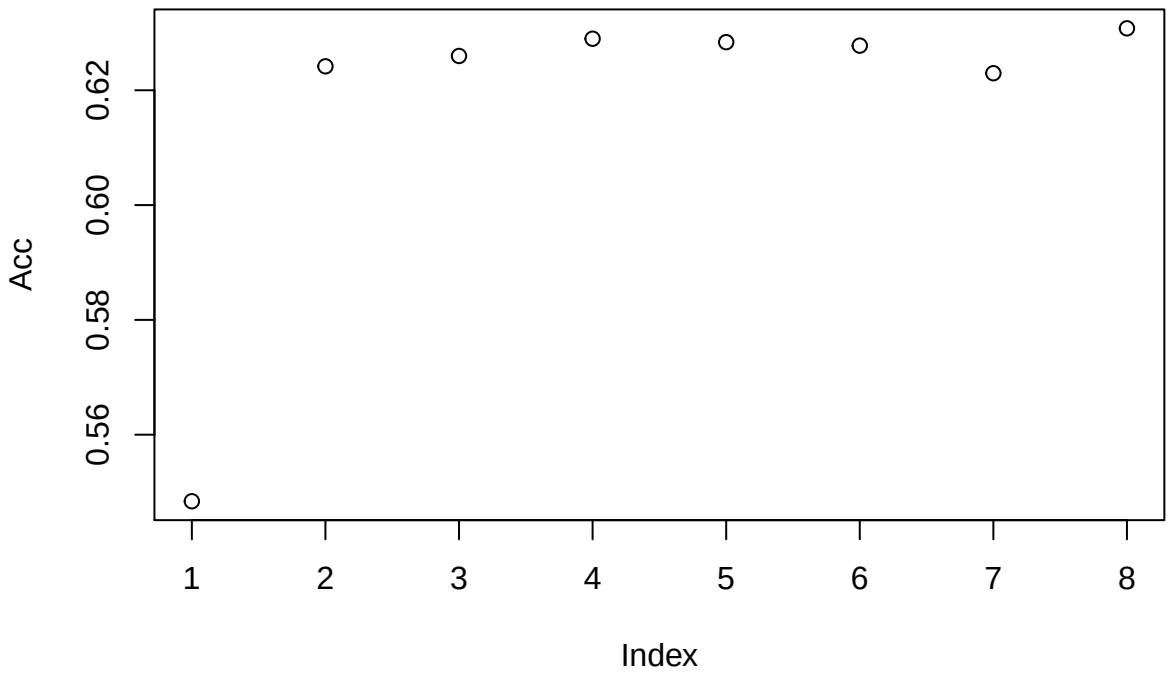
```
# Tune ntree parameter
Acc=matrix(ncol=1, nrow=4000)
for (i in seq(1,4000,by=500)){
  rf=randomForest(Party ~ ., data = train1, mtry=105, ntree=i)
  pred=predict(rf, newdata=train2)
  tab=table(train2$Party, pred)
  Acc[i]=sum(diag(tab))/sum(tab)

}
Acc <- Acc[!is.na(Acc)]
max(Acc)
```

```
## [1] 0.6307877
```

```
which.max(Acc)
```

```
## [1] 8
```

```
plot(Acc)
```



Using these two parameter values, we construct the Random Forest model using all predictors:

```
rf2=randomForest(Party ~ ., data = imptrain, mtry=105, ntree=3501)

rfpred2=predict(rf2, newdata=imptest)

rfm2 = data.frame(USER_ID = test$USER_ID, Predictions = rfpred2)

write.csv(rfm2, "rfm2.csv", row.names=FALSE)
```

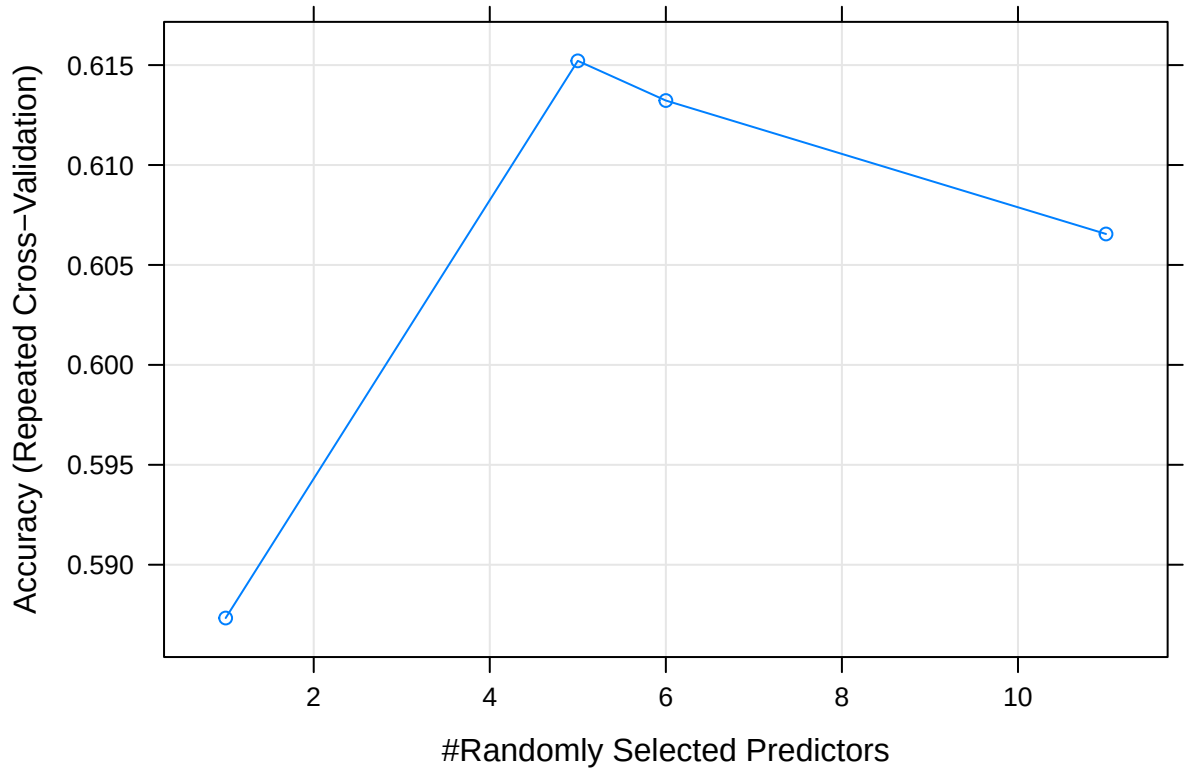
This model gives us a score of 0.63937 on the private leaderboard. Now we will try a simpler 4-predictor model:

```
# Tune mtry parameter
control <- trainControl(method="repeatedcv", number=5, repeats=2, search="random")
set.seed(1999)
mtry <- sqrt(ncol(imptrain))
rf_random <- train(Party ~ Q109244 + Q113181 + Q115611 + Income, data=imptrain, method="rf", tuneLength=
print(rf_random)
```

```
## Random Forest
##
## 5542 samples
##  106 predictor
##    2 classes: 'Democrat', 'Republican'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 2 times)
## Summary of sample sizes: 4433, 4433, 4434, 4434, 4434, 4433, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.5873326  0.1399518
##    5    0.6152116  0.2201520
##    6    0.6132270  0.2164645
##   11    0.6065495  0.2043109
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 5.
```

```
plot(rf_random)
```




```
rf3=randomForest(Party ~ Q109244 + Q113181 + Q115611 + Income, data = imptrain, mtry=5, ntree=3501)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range
```

```
rfpred3=predict(rf3, newdata=imptest)
```

```
rfm3 = data.frame(USER_ID = test$USER_ID, Predictions = rfpred3)

write.csv(rfm3, "rfm3.csv", row.names=FALSE)
```

However, this model only obtains 0.63075 accuracy on the private leaderboard.

Conclusion

The best we could do was an accuracy of 0.63937 on the private leaderboard, with both Random Forest and logistic regression models. This places us in the top 10 percent of the competition.

Additional ideas for further improving accuracy: cluster-then-predict, additional types of models such as gradient boosting, imputing the demographic variables with MICE instead of filling in with median values.