STP598 Final Project: Predicting 2016 Election Vote Choices

Xin Liu (alphabetical)
Fan Hong
Dantong Huang
Hao Wang
Arizona State University

May 04, 2017

Contents

1 Description of our study

In this study we want to predict vote choice in the recent 2016 presidential election based on a high-quality survey data: Cooperative Congressional Election Study click here. Starting in 2006, CCES is a combined effort of 39 universities. The study has continued every year since. Then joint efforts have produced national sample surveys in excess of 50,000 respondents in every federal election since. Professors Stephen Ansolabehere of Harvard

University and Brian Schaffner of the University of Massachusetts coordinate the CCES and YouGov in Palo Alto, CA, conducts and distributes the surveys.

In our group project, we use the recent-released 2016 CCES, which includes 64600 observations in total. Due to the missingness of some variables, we ended up with around 35000 observations. This sample size is large enough to achieve consistent and stable estimates. We have several goals in mind:

1. Variable slection: Find out the most influential predictors among all the possible variables
2. Model prediction: Find out the method with the highest correct-prediction rates.

2 Variable Selection and Data Cleaning

Based on our previous knowledge, we prepared dataset based on two parts. The first part includes all the possible demographic predictors including gender, race, marriage status, state, religion, occupation, employment status, home ownership, family income. In the second part we include predictors involving partisanship and other political attitude variables: party ID, evaluation of national economy, evaluation on security, on healthcare, on trade policies, on defence and security, as well as approval ratings of the House, Senate, the Court and the former President Obama. Total we include 36 variables.

We did multiple approaches to clean the data and make it appropriate for analyses. First, since we only look at vote choice, people who did not vote in the 2016 Election were dropped in our sample. Second, we did a series of transformation to make it suiatble for shrinkage regression: the original dataset is a survey data, each question is read by R as `ordered factors`. Some of them like `state`, `race`, `religion` are apparently not ordered, but just categorical. We coded these variables as `factor`.

We lost about 8000 observations due to the **not voted** option. Besides, we lost another 6000 observations in the question of family income: these individuals choose **prefer not to answer** option in this question. To simplify our prediction outcomes, we ignored those people who voted for Jill Stein, Gary Johnson or other minority candidates, there are about 2000 people in the survey who voted for them.

```
#read data
load(url("https://dataverse.harvard.edu/api/access/datafile/3004425?format=RData&gbrecs=

library(dplyr)
library(glmnet)
library(pROC)
library(rpart)
library(randomForest)
library(xgboost)
```

```r
mydata <- dplyr::select(x,
                        CC16_410a, #vote for
                        gender, #gender
                        educ, #education
                        race, #race
                        marstat, #marriage
                        inputstate, #state
                        religpew, #religion
                        industryclass, #industry class
                        ownhome, #home owner
                        immstat, #immigrant
                        faminc, #family income
                        employ, #employment status
                        pid7, #party id
                        CC16_302, #national econ better: past
                        CC16_303, #homehold income better: past
                        CC16_304, # national econ better: next
                        CC16_307, # feel safe about police
                        CC16_321a,
                        CC16_321b,
                        CC16_320a,
                        CC16_320b,
                        CC16_331_1,
                        CC16_331_2,
                        CC16_331_3,
                        CC16_331_7,
                        CC16_332a,
                        CC16_332b,
                        CC16_332c,
                        CC16_332f,
                        CC16_334c,
                        CC16_334d,
                        CC16_335,
                        CC16_337_1,
                        CC16_337_2,
                        CC16_337_3)

#select only trump and clinton
mydata <- mydata %>%
          filter(CC16_410a %in%
          c("Donald Trump (Republican)",
            "Hillary Clinton (Democrat)"))

#relevel Trump 0 Clinton 1
```

```r
mydata$CC16_410a <- as.numeric(mydata$CC16_410a) - 1
mydata$CC16_410a <- as.factor(mydata$CC16_410a)


#covert ordered to factor
mydata$race <- factor(mydata$race, ordered = FALSE)
mydata$inputstate <- factor(mydata$inputstate, ordered = FALSE)
mydata$religpew <- factor(mydata$religpew, ordered = FALSE)
mydata$inputstate <- factor(mydata$inputstate, ordered = FALSE)
mydata$industryclass <- factor(mydata$industryclass, ordered = FALSE)
mydata$faminc[mydata$faminc == "Prefer not to say"] <- NA


mydata <- na.omit(mydata)
```

3 LASSO

We first conduct our analysis with LASSO. LASSO requires additional transformation of this dataset. Since L1 regularization cannot handle factor data, we did several additional steps:

1. We transform the categorical variables with k levels like state into k-1 dummies
2. We convert the ordered factor variables like income into numeric variables.

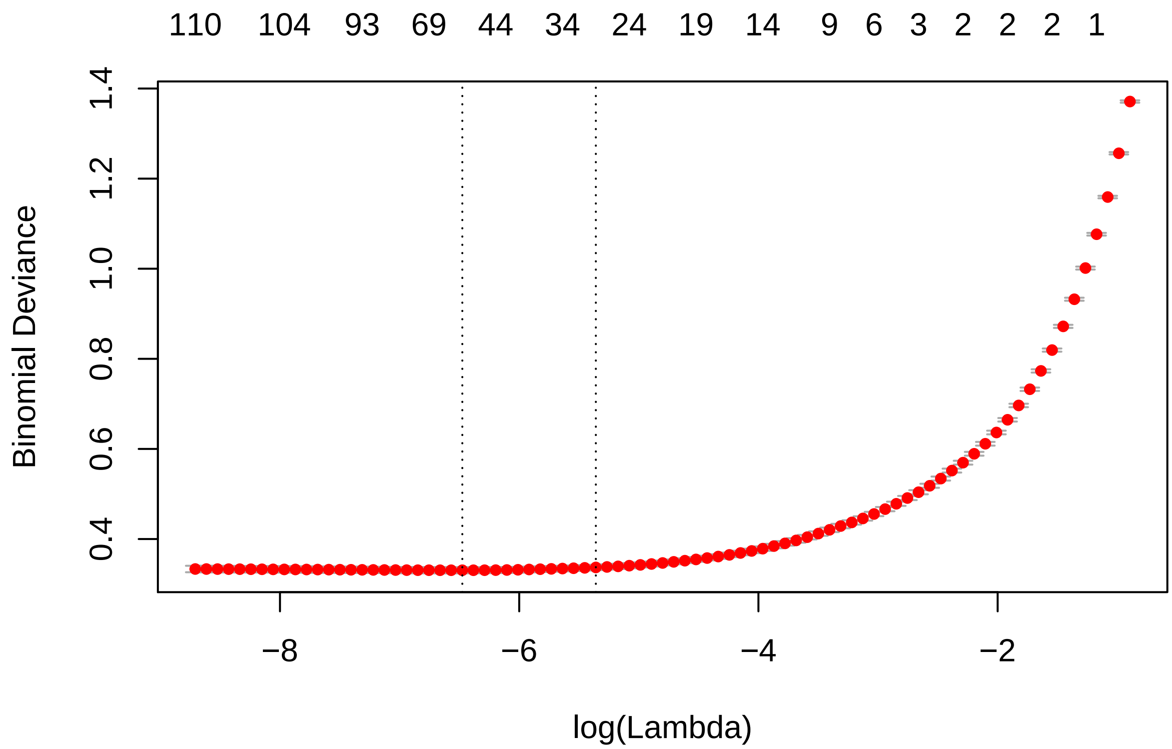
3.1 Data Cleaning

```r
analysis3 <- mydata %>%
             select(race,
                inputstate,
                religpew,
                industryclass)

analysis4 <- mydata %>%
             select(-race,
                -inputstate,
                -religpew,
                -industryclass)
#coerce to numeric values
analysis4 <- as.data.frame(sapply(analysis4, as.numeric))

############################################################
```

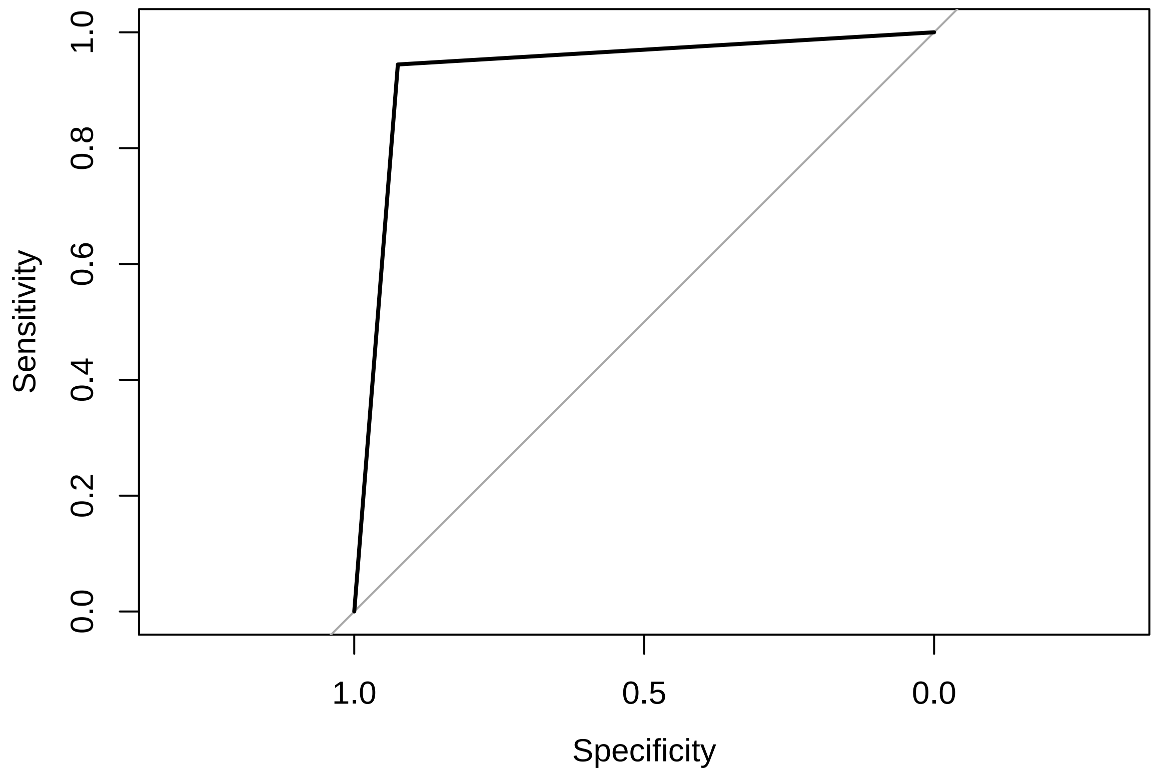
```
## dummies for race, inputstate, religpew, industryclass, R automatically expands factor
fm = as.formula(~.)
xm <- model.matrix(fm, analysis3)
xm <- as.data.frame(xm)
xm <- dplyr::select(xm, -1)
lasso.data <- cbind.data.frame(analysis4, xm)
```

3.2 Model Evaluation

```
#data split
n = nrow(lasso.data)
n1 = floor(n/3)
data.test = sample_n(lasso.data, n1)
data.train  = setdiff(lasso.data, data.test)

#lasso train
set.seed(99)
y <- as.factor(data.train$CC16_410a) #Trump 0 Clinton 1
x <- as.matrix(data.train[, 2:ncol(lasso.data)])

cvfit <- cv.glmnet(x,
                   y,
                   nfolds = 10, #10 fold
                   family = "binomial",
                   alpha = 1) #Lasso

plot(cvfit)
```




```
x.test <- as.matrix(data.test[, 2:ncol(data.test)])
lasso.pred <- predict(cvfit, newx = x.test, s = "lambda.min", type = "class")
lasso.pred <- as.numeric(lasso.pred)

rocCurve = pROC::roc(response = data.test$CC16_410a, predictor = lasso.pred)
plot(rocCurve)
```



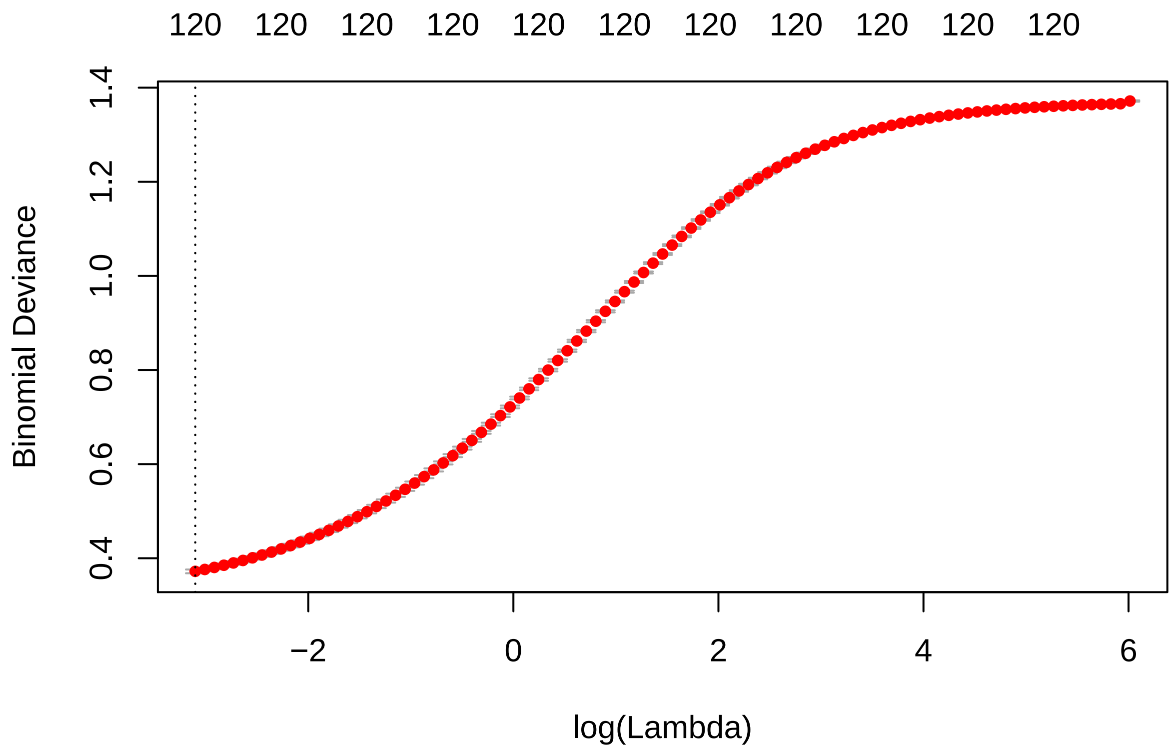

```r
cat("auc LASSO", auc(rocCurve),"\n")
```

```
## auc LASSO 0.9346333
```

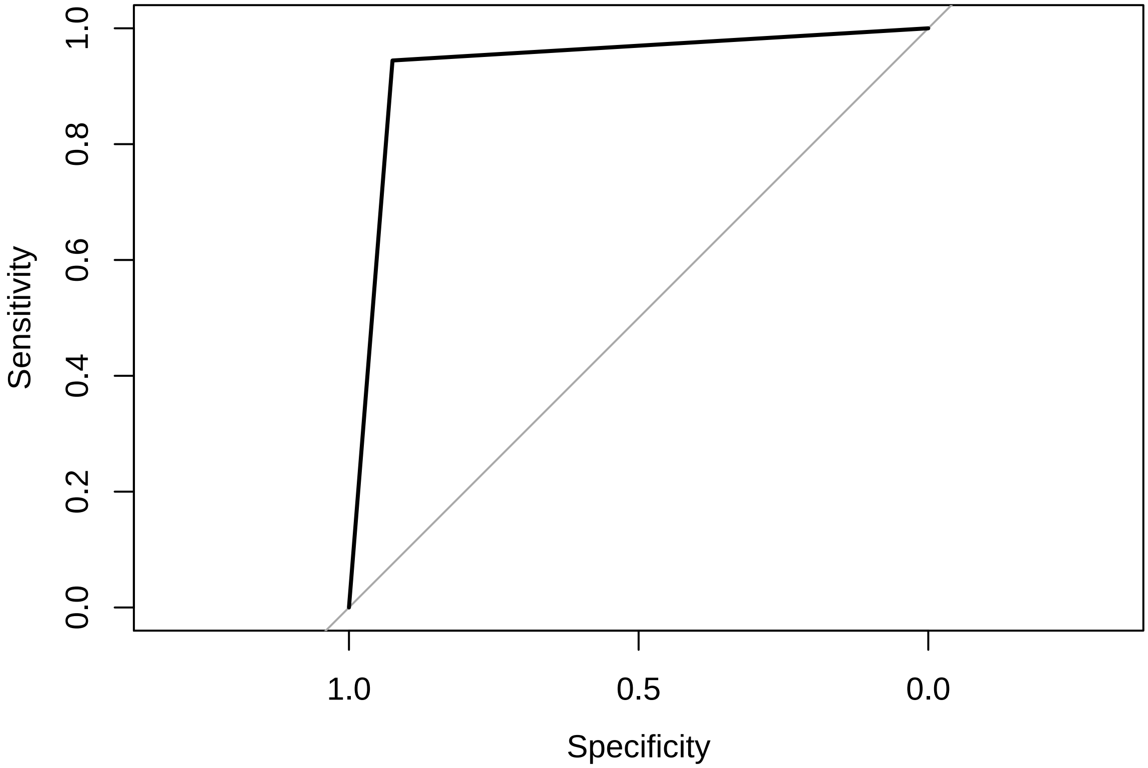
4 Ridge

The codes between ridge and LASSO are very similar, all we need to change is the alpha option

```r
cvfit <- cv.glmnet(x,
                  y,
                  nfolds = 10, #10 fold
                  family = "binomial",
                  alpha = 0) #Ridge

plot(cvfit)
```




```r
ridge.pred <- predict(cvfit, newx = x.test, s = "lambda.min", type = "class")
ridge.pred <- as.numeric(lasso.pred)


rocCurve = roc(response = data.test$CC16_410a, predictor = ridge.pred)
plot(rocCurve)
```




```
cat("auc  Ridge", auc(rocCurve),"\n")
```

```
## auc  Ridge 0.9346333
```

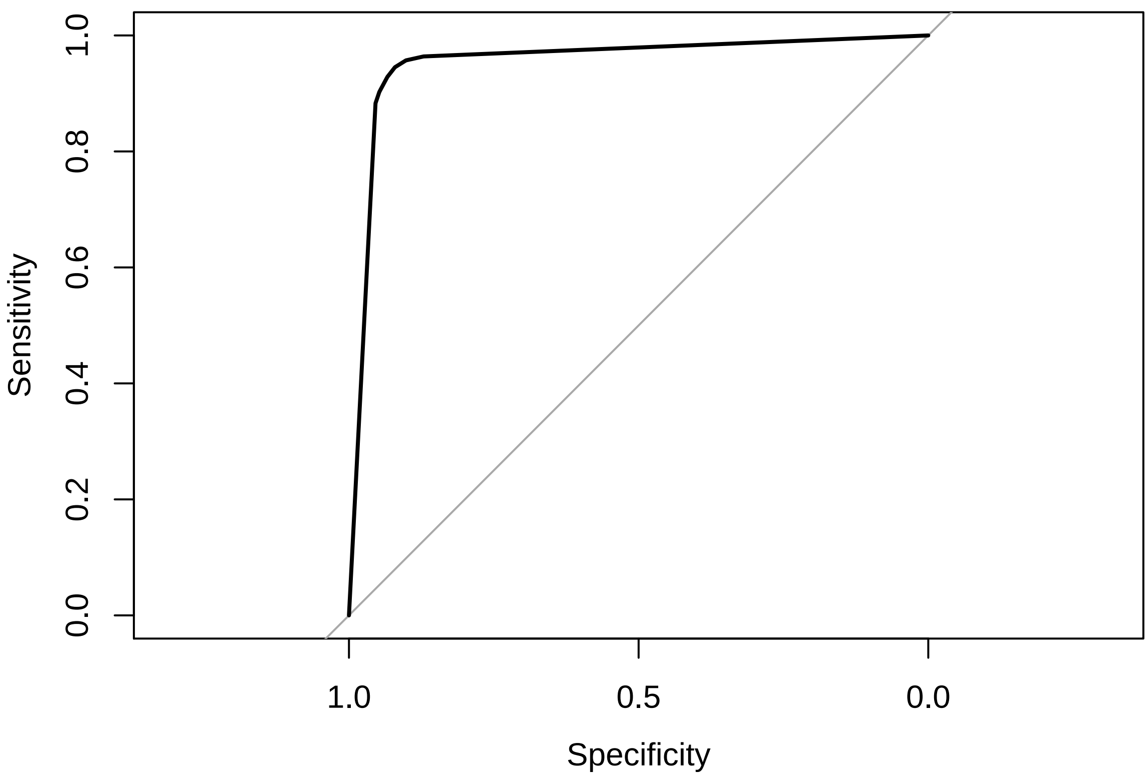
5 Tree-Based Method

```
#data split
n <- nrow(mydata)
n1 <- floor(n/3)
data.test <- sample_n(mydata, n1)
data.train  <- setdiff(mydata, data.test)

set.seed(99)
big.tree = rpart(CC16_410a ~ .,
                 method = "class",
                 data = data.train,
                 control = rpart.control(minsplit = 50, cp = .0001))

nbig = length( unique(big.tree$where) )
```

```r
cat("size of big tree: ", nbig, "\n")
```

```
## size of big tree:  62
```

```r
iibest = which.min(big.tree$cptable[,"xerror"]) #which has the lowest error
bestcp = big.tree$cptable[iibest,"CP"]
bestsize = big.tree$cptable[iibest,"nsplit"] + 1
best.tree = prune(big.tree,cp = bestcp)

tree.pred = predict(best.tree,newdata = data.test,type = "prob")[,2]

rocCurve = roc(response = data.test$CC16_410a,predictor = tree.pred)
plot(rocCurve)
```




```r
cat("auc, tree: ", auc(rocCurve),"\n")
```

```
## auc, tree:  0.9538206
```

6 Random Forest

```
#data split
n <- nrow(mydata)
n1 <- floor(n/3)
data.test <- sample_n(mydata, n1)
data.train  <- setdiff(mydata, data.test)

set.seed(99)
p = ncol(data.train) - 1
mtryv = c(p,sqrt(p))
ntreev = c(4,16,32,64,128,256)
setrf = expand.grid(mtryv,ntreev)
colnames(setrf) = c("mtry","ntree")
rf = matrix(0.0,nrow(data.test),nrow(setrf))

###fit rf

for (i in 1:nrow(setrf)) {
cat("on randomForest fit ", i, "\n")
print(setrf[i,])
#fit and predict
frf = randomForest(CC16_410a ~. ,
                   data = data.train,
                   mtry = setrf[i,1],
                   ntree = setrf[i,2])
phat = predict(frf,
               newdata = data.test,
               type = "prob")[,2]
rf[,i] = phat
}
```

```
## on randomForest fit  1
##   mtry ntree
## 1   34     4
## on randomForest fit  2
##       mtry ntree
## 2 5.830952     4
## on randomForest fit  3
##   mtry ntree
## 3   34    16
## on randomForest fit  4
##       mtry ntree
## 4 5.830952    16
```

```
## on randomForest fit  5
##   mtry ntree
## 5   34    32
## on randomForest fit  6
##       mtry ntree
## 6 5.830952    32
## on randomForest fit  7
##   mtry ntree
## 7   34    64
## on randomForest fit  8
##       mtry ntree
## 8 5.830952    64
## on randomForest fit  9
##   mtry ntree
## 9   34   128
## on randomForest fit  10
##        mtry ntree
## 10 5.830952   128
## on randomForest fit  11
##    mtry ntree
## 11   34   256
## on randomForest fit  12
##        mtry ntree
## 12 5.830952   256
```

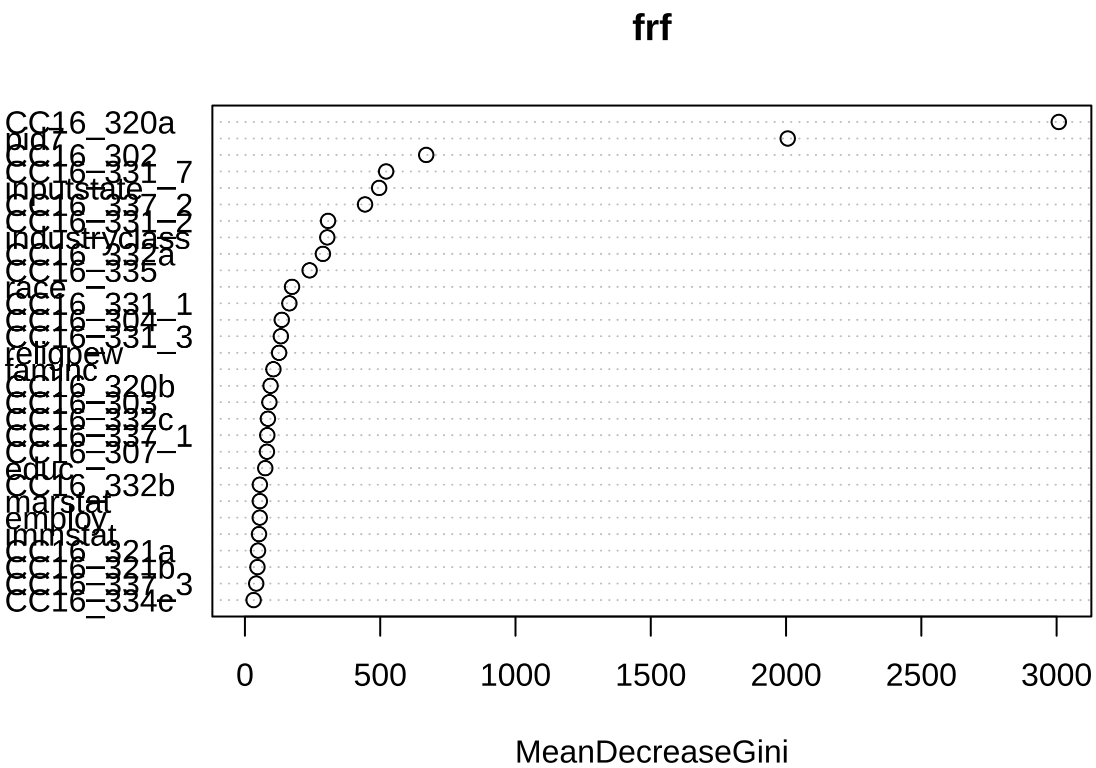
```
for (i in 1:ncol(rf)) {
  rocCurve = roc(response = data.test$CC16_410a,
                 predictor = rf[,i])
  cat("auc, random forest ", i,": ", auc(rocCurve),"\n")
}
```

```
## auc, random forest  1 :  0.958653
## auc, random forest  2 :  0.9630837
## auc, random forest  3 :  0.9776551
## auc, random forest  4 :  0.9798002
## auc, random forest  5 :  0.9807524
## auc, random forest  6 :  0.984214
## auc, random forest  7 :  0.9825937
## auc, random forest  8 :  0.9861089
## auc, random forest  9 :  0.9836334
## auc, random forest  10 :  0.9866741
## auc, random forest  11 :  0.9836128
## auc, random forest  12 :  0.9866534
```

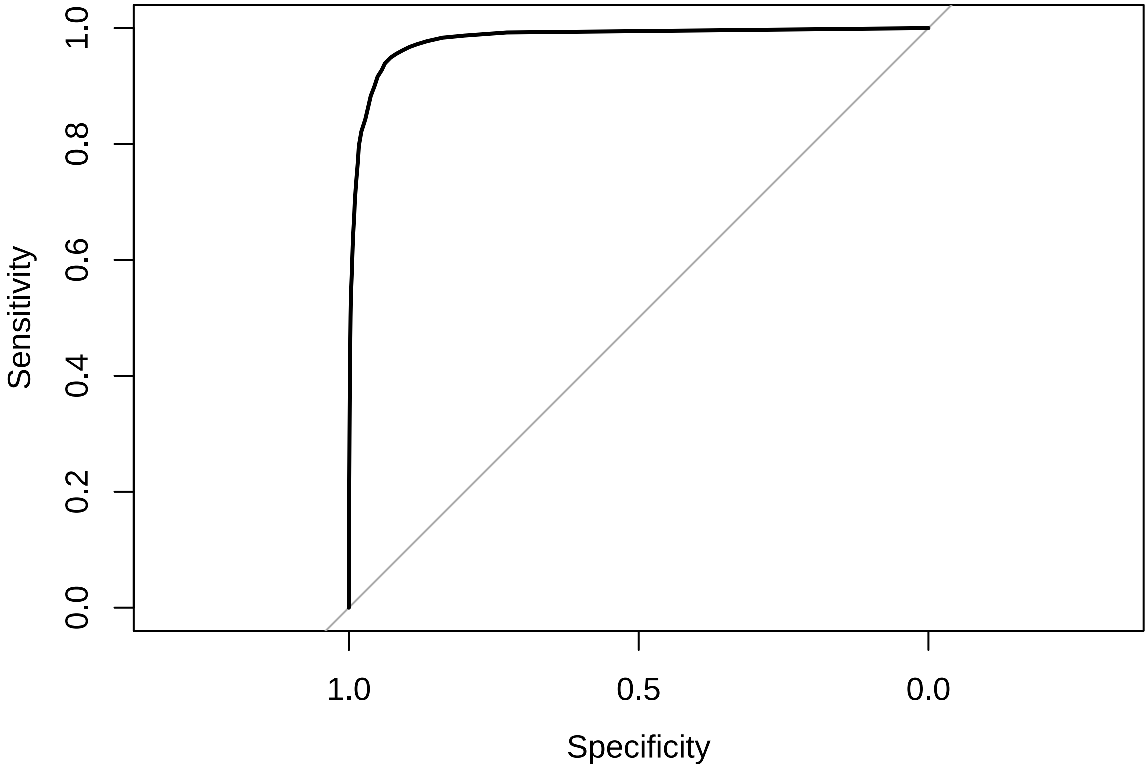
```r
importance(frf)
```

```
##                 MeanDecreaseGini
## gender                  30.18404
## educ                    75.05545
## race                   174.07077
## marstat                 55.12283
## inputstate             496.03140
## religpew               126.28269
## industryclass          304.36443
## ownhome                 31.23734
## immstat                 52.06594
## faminc                 105.24288
## employ                  54.82390
## pid7                  2006.20388
## CC16_302               669.99718
## CC16_303                90.39677
## CC16_304               136.22825
## CC16_307                81.25791
## CC16_321a               48.34237
## CC16_321b               46.22267
## CC16_320a             3008.11658
## CC16_320b               94.67543
## CC16_331_1             164.23110
## CC16_331_2             307.23878
## CC16_331_3             132.70736
## CC16_331_7             521.90827
## CC16_332a              288.15368
## CC16_332b               55.19880
## CC16_332c               84.79307
## CC16_332f               28.29953
## CC16_334c               32.15668
## CC16_334d               20.90936
## CC16_335               239.12835
## CC16_337_1              82.59642
## CC16_337_2             443.88068
## CC16_337_3              41.69121
```

```r
library(caret)
varImpPlot(frf,type = 2)
```




```r
# The 5th rf fit
rocCurve = roc(response = data.test$CC16_410a, predictor = rf[, 5])
plot(rocCurve)
```



7 Boosting

```
##settings for boosting
idv = c(2,4)
ntv = c(1000,5000)
shv = c(.1,.01)
setboost = expand.grid(idv, ntv, shv)
colnames(setboost) = c("tdepth","ntree","shrink")
boost = matrix(0.0,
               nrow(data.test),
               nrow(setboost))

trainDfB = data.train
trainDfB$CC16_410a = as.numeric(data.train$CC16_410a) - 1

testDfB = data.test
testDfB$CC16_410a = as.numeric(data.test$CC16_410a) - 1

##fit boosting
```

```r
library(gbm)
```

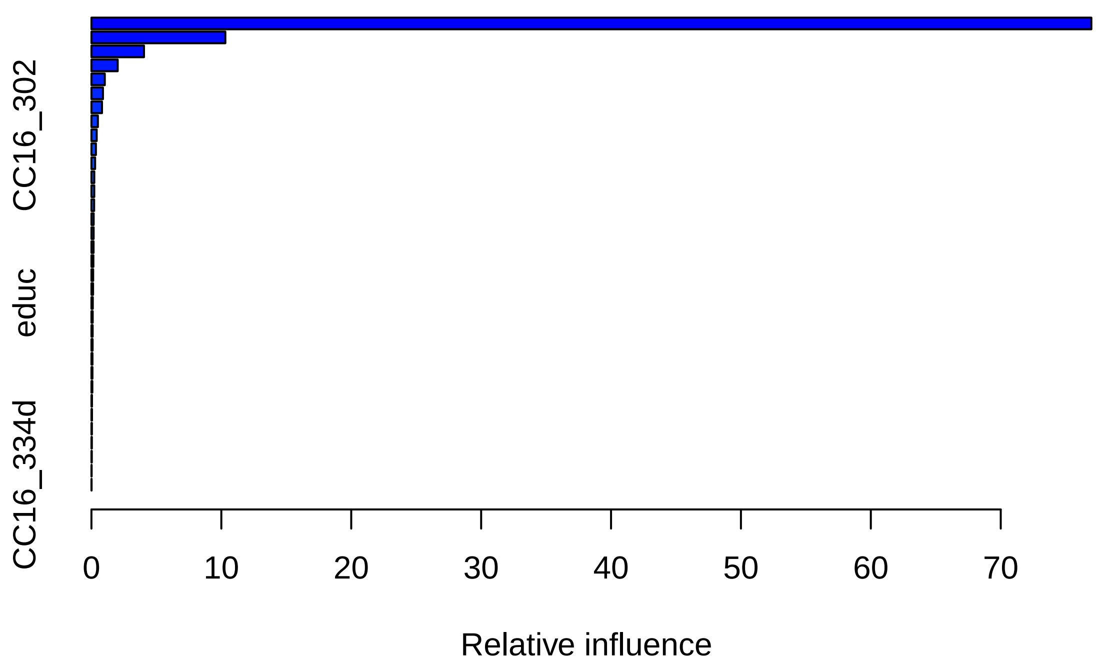
```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```r
tmboost = system.time({#get the time, will use this later
  for (i in 1:nrow(setboost)) {
    cat("on boosting fit ",i,"\n")
    print(setboost[i,])
    ##fit and predict
    fboost = gbm(CC16_410a ~.,
                 data = trainDfB,
                 distribution = "bernoulli",
                 n.trees = setboost[i,2],
                 interaction.depth = setboost[i,1],
                 shrinkage = setboost[i,3])
    phat = predict(fboost,
                   newdata = testDfB,
                   n.trees = setboost[i,2],
                   type = "response")
    boost[,i] = phat
  }
})
```

```
## on boosting fit  1
##   tdepth ntree shrink
## 1      2  1000    0.1
## on boosting fit  2
##   tdepth ntree shrink
## 2      4  1000    0.1
## on boosting fit  3
##   tdepth ntree shrink
## 3      2  5000    0.1
## on boosting fit  4
##   tdepth ntree shrink
## 4      4  5000    0.1
```

```
## on boosting fit  5
##   tdepth ntree shrink
## 5      2  1000   0.01
## on boosting fit  6
##   tdepth ntree shrink
## 6      4  1000   0.01
## on boosting fit  7
##   tdepth ntree shrink
## 7      2  5000   0.01
## on boosting fit  8
##   tdepth ntree shrink
## 8      4  5000   0.01
```

```r
summary(fboost)
```




```
##                              var      rel.inf
## CC16_320a          CC16_320a 76.97666351
## pid7                    pid7 10.31037118
## inputstate        inputstate  4.04567933
## industryclass industryclass  2.02296297
## CC16_320b          CC16_320b  1.02835004
## race                    race  0.88660521
```

```
## CC16_331_7       CC16_331_7  0.81289338
## religpew           religpew  0.49692739
## CC16_302           CC16_302  0.40524128
## CC16_331_2       CC16_331_2  0.34418176
## CC16_337_2       CC16_337_2  0.28316397
## CC16_332a         CC16_332a  0.21904456
## CC16_335           CC16_335  0.21804479
## CC16_321a         CC16_321a  0.20634759
## CC16_307           CC16_307  0.17254498
## CC16_332f         CC16_332f  0.17101360
## CC16_331_3       CC16_331_3  0.16468818
## CC16_321b         CC16_321b  0.14956089
## CC16_331_1       CC16_331_1  0.13388631
## immstat             immstat  0.12422888
## educ                   educ  0.10378355
## CC16_332b         CC16_332b  0.09833181
## CC16_332c         CC16_332c  0.09371993
## faminc               faminc  0.09107708
## CC16_303           CC16_303  0.07973754
## CC16_304           CC16_304  0.07752034
## gender               gender  0.06987075
## marstat             marstat  0.04344673
## employ               employ  0.04338667
## ownhome             ownhome  0.03663195
## CC16_337_3       CC16_337_3  0.03228824
## CC16_337_1       CC16_337_1  0.02862657
## CC16_334c         CC16_334c  0.01741392
## CC16_334d         CC16_334d  0.01176511
```

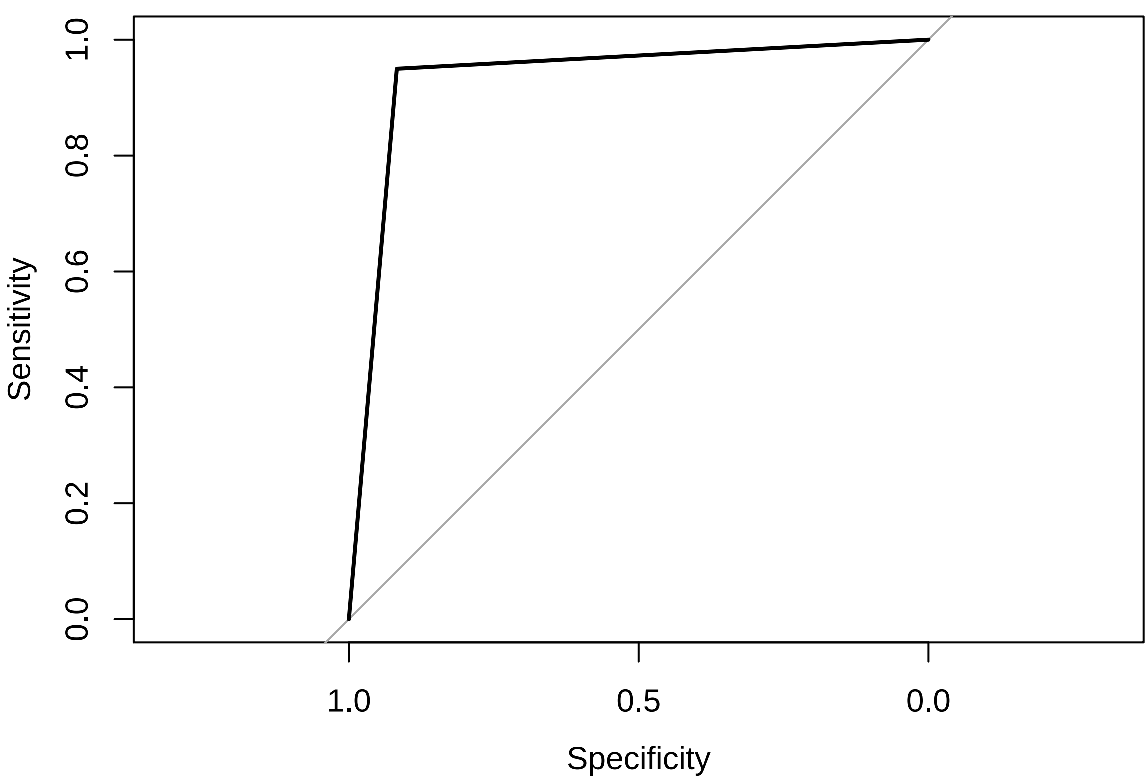
```
for (n in 1:ncol(boost)){
  rocCurve = roc(response = data.test$CC16_410a,
                 predictor = boost[,n])
  cat("auc, boost ",n,": ", auc(rocCurve),"\n")
}
```

```
## auc, boost  1 :  0.9863262
## auc, boost  2 :  0.9850165
## auc, boost  3 :  0.982901
## auc, boost  4 :  0.9814362
## auc, boost  5 :  0.9867182
## auc, boost  6 :  0.987258
## auc, boost  7 :  0.9872172
## auc, boost  8 :  0.9870131
```

8 GLM

Based on the random forest most important variables, we use the basic GLM function to check how better other methods are

```r
#data split
library(dplyr)
n = nrow(mydata)
n1 = floor(n/3)
data.train = sample_n(mydata, n1)
data.test  = setdiff(mydata, data.train)

glm <- glm(CC16_410a ~ CC16_320a + pid7 , data = data.train, family = "binomial" )
glm.pred <- predict(glm, newdata = data.test, type = "response")
glm.pred <- as.numeric(glm.pred > 0.5)

rocCurve2 = roc(response = data.test$CC16_410a, predictor = glm.pred)
plot(rocCurve2)
```




```r
library(caret)
confusionMatrix(data.test$CC16_410a, glm.pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0  8307   749
##          1   578 10943
##
##                Accuracy : 0.9355
##                  95% CI : (0.9321, 0.9388)
##     No Information Rate : 0.5682
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8689
##  Mcnemar's Test P-Value : 3.06e-06
##
##             Sensitivity : 0.9349
##             Specificity : 0.9359
##          Pos Pred Value : 0.9173
##          Neg Pred Value : 0.9498
##              Prevalence : 0.4318
##          Detection Rate : 0.4037
##    Detection Prevalence : 0.4401
##       Balanced Accuracy : 0.9354
##
##        'Positive' Class : 0
##
```

```r
cat("auc GLM", auc(rocCurve2),"\n")
```

```
## auc GLM 0.9335616
```

9 xgboost

```r
mydata$CC16_410a <- as.numeric(mydata$CC16_410a) - 1

### Data Splitting ###
set.seed(99)
n = nrow(mydata)
n1 = floor(n/2)
n2 = floor(n/4)
n3 = n - n1 - n2
ii = sample(1:n)
Train = mydata[ii[1:n1],]
```

```r
Valid = mydata[ii[n1+1:n2],]
Test =  mydata[ii[n1+n2+1:n3],]

library("xgboost")
require(Matrix)

TrainLabel <- Train$CC16_410a
TrainSparseData <- sparse.model.matrix(CC16_410a~.-1, data = Train)
TrainSparseData <- TrainSparseData[,colSums(TrainSparseData!=0)!=0]
xgTrain <- xgb.DMatrix(data = TrainSparseData, label=TrainLabel)

ValidLabel <- Valid$CC16_410a
ValidSparseData <- sparse.model.matrix(CC16_410a~.-1, data = Valid)
ValidSparseData <- ValidSparseData[,colSums(ValidSparseData!=0)!=0]
xgValid <- xgb.DMatrix(data=ValidSparseData, label=ValidLabel)

watchlist <- list(train=xgTrain,test=xgValid)
param <- list("objective" = "binary:logistic",
              "eta" = 0.1,
              "gamma" = 0.2,
              "min_child_weight" = 5,
              "subsample" = .8,
              "colsample_bytree" = .8,
              "scale_pos_weight" = 1.0,
              "verbose" = T,
              "max_depth" = 4
)

nrounds <- 500
tmxgboost = system.time({
bst <- xgb.train(params=param, data=xgTrain,
                 nrounds=nrounds, watchlist=watchlist, early_stopping_rounds=50)

TestLabel <- Test$CC16_410a
TestSparseData <- sparse.model.matrix(CC16_410a~.-1, data = Test)
TestSparseData <- TestSparseData[,colSums(TestSparseData!=0)!=0]

pred <- predict(bst, TestSparseData)
prediction <- as.numeric(pred>0.5)
err <- mean(prediction != TestLabel)
cat("error with xgboost is ", err)
})
```

```
## [1]  train-error:0.061083    test-error:0.065682
```

```
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 50 rounds.
##
## [2]  train-error:0.061277    test-error:0.065941
## [3]  train-error:0.059982    test-error:0.064257
## [4]  train-error:0.059593    test-error:0.063480
## [5]  train-error:0.059010    test-error:0.062573
## [6]  train-error:0.059010    test-error:0.063480
## [7]  train-error:0.059010    test-error:0.064257
## [8]  train-error:0.058557    test-error:0.063998
## [9]  train-error:0.057715    test-error:0.063739
## [10] train-error:0.057326    test-error:0.063998
## [11] train-error:0.057391    test-error:0.063739
## [12] train-error:0.057456    test-error:0.063868
## [13] train-error:0.057456    test-error:0.063350
## [14] train-error:0.057456    test-error:0.063221
## [15] train-error:0.057585    test-error:0.063480
## [16] train-error:0.057715    test-error:0.063350
## [17] train-error:0.057261    test-error:0.063350
## [18] train-error:0.056937    test-error:0.062702
## [19] train-error:0.056873    test-error:0.062962
## [20] train-error:0.056808    test-error:0.062832
## [21] train-error:0.056160    test-error:0.062314
## [22] train-error:0.055707    test-error:0.061796
## [23] train-error:0.055512    test-error:0.061796
## [24] train-error:0.055383    test-error:0.061666
## [25] train-error:0.055059    test-error:0.061925
## [26] train-error:0.054476    test-error:0.061925
## [27] train-error:0.054411    test-error:0.061666
## [28] train-error:0.054346    test-error:0.061666
## [29] train-error:0.054411    test-error:0.061536
## [30] train-error:0.054087    test-error:0.061277
## [31] train-error:0.053245    test-error:0.061536
## [32] train-error:0.053245    test-error:0.061407
## [33] train-error:0.052921    test-error:0.061796
## [34] train-error:0.052662    test-error:0.060889
## [35] train-error:0.052533    test-error:0.060241
## [36] train-error:0.052403    test-error:0.060371
## [37] train-error:0.052209    test-error:0.060241
## [38] train-error:0.052533    test-error:0.060111
## [39] train-error:0.052209    test-error:0.060500
## [40] train-error:0.051885    test-error:0.060241
## [41] train-error:0.052079    test-error:0.060500
## [42] train-error:0.051561    test-error:0.060241
## [43] train-error:0.051626    test-error:0.059982
```

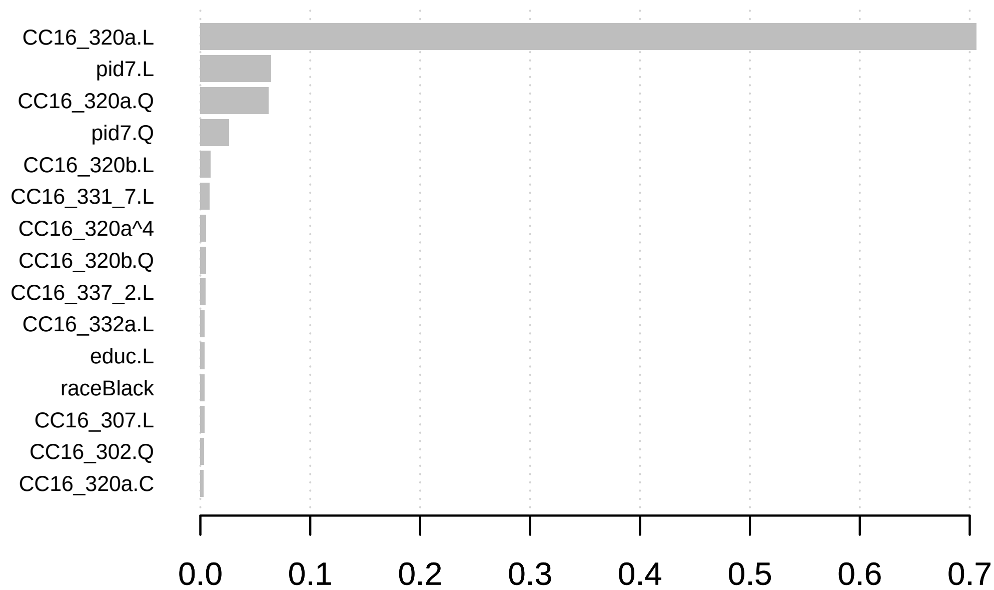
```
## [44] train-error:0.051237    test-error:0.059852
## [45] train-error:0.051432    test-error:0.059852
## [46] train-error:0.051108    test-error:0.059593
## [47] train-error:0.051043    test-error:0.059464
## [48] train-error:0.050719    test-error:0.059464
## [49] train-error:0.051172    test-error:0.059334
## [50] train-error:0.050849    test-error:0.058945
## [51] train-error:0.050460    test-error:0.058945
## [52] train-error:0.050589    test-error:0.058945
## [53] train-error:0.050784    test-error:0.059334
## [54] train-error:0.050330    test-error:0.059982
## [55] train-error:0.050071    test-error:0.059593
## [56] train-error:0.050071    test-error:0.059334
## [57] train-error:0.050136    test-error:0.059464
## [58] train-error:0.049812    test-error:0.059982
## [59] train-error:0.049488    test-error:0.060241
## [60] train-error:0.049424    test-error:0.060241
## [61] train-error:0.049229    test-error:0.060241
## [62] train-error:0.049035    test-error:0.060371
## [63] train-error:0.048711    test-error:0.060500
## [64] train-error:0.048517    test-error:0.060371
## [65] train-error:0.048258    test-error:0.059852
## [66] train-error:0.047998    test-error:0.059852
## [67] train-error:0.047804    test-error:0.060111
## [68] train-error:0.048128    test-error:0.059982
## [69] train-error:0.048452    test-error:0.060371
## [70] train-error:0.047739    test-error:0.060111
## [71] train-error:0.047675    test-error:0.060241
## [72] train-error:0.047221    test-error:0.059852
## [73] train-error:0.047286    test-error:0.059852
## [74] train-error:0.047221    test-error:0.060111
## [75] train-error:0.047156    test-error:0.059723
## [76] train-error:0.047092    test-error:0.059982
## [77] train-error:0.046962    test-error:0.059852
## [78] train-error:0.046897    test-error:0.059852
## [79] train-error:0.046768    test-error:0.059982
## [80] train-error:0.046703    test-error:0.059852
## [81] train-error:0.046509    test-error:0.059982
## [82] train-error:0.046509    test-error:0.060111
## [83] train-error:0.046444    test-error:0.059982
## [84] train-error:0.046638    test-error:0.059852
## [85] train-error:0.046573    test-error:0.059852
## [86] train-error:0.046444    test-error:0.059852
## [87] train-error:0.046509    test-error:0.059723
## [88] train-error:0.046185    test-error:0.060241
```

```
## [89] train-error:0.046185    test-error:0.060111
## [90] train-error:0.045990    test-error:0.060371
## [91] train-error:0.045667    test-error:0.059982
## [92] train-error:0.045731    test-error:0.059982
## [93] train-error:0.045472    test-error:0.059852
## [94] train-error:0.045472    test-error:0.059464
## [95] train-error:0.045148    test-error:0.059593
## [96] train-error:0.045148    test-error:0.059982
## [97] train-error:0.044889    test-error:0.059723
## [98] train-error:0.044889    test-error:0.059593
## [99] train-error:0.044695    test-error:0.059723
## [100]    train-error:0.044630    test-error:0.059593
## Stopping. Best iteration:
## [50] train-error:0.050849    test-error:0.058945
##
## error with xgboost is  0.05985231
```

```r
top_n = 15
importance_matrix <- xgb.importance(colnames(TrainSparseData), model = bst)
print(importance_matrix[1:top_n,])
```

```
##          Feature        Gain       Cover   Frequency
##  1:  CC16_320a.L 0.706018405 0.152011374 0.039611360
##  2:       pid7.L 0.064410115 0.129370868 0.062780269
##  3:  CC16_320a.Q 0.062311081 0.066049434 0.024663677
##  4:       pid7.Q 0.026135595 0.044010970 0.017937220
##  5:  CC16_320b.L 0.009349014 0.052275949 0.032884903
##  6: CC16_331_7.L 0.008442786 0.034339273 0.026158445
##  7:  CC16_320a^4 0.005364449 0.009550486 0.006726457
##  8:  CC16_320b.Q 0.005137694 0.022799998 0.018684604
##  9: CC16_337_2.L 0.004982552 0.017015537 0.017937220
## 10:  CC16_332a.L 0.003730190 0.015443025 0.015695067
## 11:       educ.L 0.003688553 0.008774904 0.017189836
## 12:    raceBlack 0.003662215 0.027083573 0.019431988
## 13:   CC16_307.L 0.003656547 0.015957165 0.019431988
## 14:   CC16_302.Q 0.003307484 0.016708584 0.014200299
## 15:  CC16_320a.C 0.002831362 0.013544378 0.008221226
```

```r
xgb.plot.importance(importance_matrix, top_n = top_n)
```



Compare `xgboost` and `boost`: `xgboost` much faster, though less accurate.

```
tmboost
```

```
##    user  system elapsed
##  683.48    0.11  684.34
```

```
tmxgboost
```

```
##    user  system elapsed
##   10.15    0.59    2.03
```