Problem Set 5

Akira Masuda

2020/3/1

```
Course: MACS30100 Perspectives on Computational Modeling (Winter 2020)
Author: Akira Masuda (ID: alakira)
```

```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
knitr::opts_chunk$set(fig.width=6,fig.height=3.4,fig.align='center')
```

```
library(knitr)
library(ggplot2)
library(tidyverse)
library(caret)
library(broom)
library(rsample)
library(patchwork)
library(corrplot)
library(dplyr)
library(ISLR)
library(ggplot2)
library(e1071)
library(h2o)
library(ROCR)
rm(list=ls())
set.seed(1100)
```

Conceptual: Cost functions for classification trees

1.

According to "ISLR: Data for an Introduction to Statistical Learning with Applications in R", the Gini index or the entropy are better to use when growing the tree since these cost functions are 'more sensitive to node purity than is the classification error rate'. On the other hand, for pruning, any of these three cost functions are fine, but 'the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal'.

Application: Predicting attitudes towards racist college professors

Estimate the models

2.

```
set.seed(110)
df_train <- read.csv('data/gss_train.csv') %>% mutate(colrac=as.factor(colrac)) %>% drop_na()
```

```r
df_test <- read.csv('data/gss_test.csv') %>% mutate(colrac=as.factor(colrac)) %>% drop_na()
```

```r
cv_10 <- trainControl(method='cv', number=10)

train_x <- as.matrix(df_train %>% select(-colrac))
mode(train_x) = 'numeric'
train_y <- df_train$colrac %>% as.factor()

test_x <- as.matrix(df_test %>% select(-colrac))
mode(test_x) = 'numeric'
test_y <- df_test$colrac %>% as.factor()
```

Logistic Regression

```r
(log_model <- train(x=train_x,
                   y=train_y,
                   method='glm',
                   trControl=cv_10))
```

```
## Generalized Linear Model
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1327, 1328, 1329, 1329, 1328, 1328, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7981094  0.5952716
```

Naive Bayes

```r
nb_search_grid <- expand.grid(fL=0:5, usekernel=c(TRUE, FALSE),
                              adjust=seq(0, 6, by=2))
(nb_model <- train(x=train_x,
                   y=train_y,
                   method='nb',
                   trControl=cv_10,
                   tuneGrid=nb_search_grid))
```

```
## Naive Bayes
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1328, 1328, 1329, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   fL  usekernel  adjust  Accuracy   Kappa
```

```
##    0   FALSE      0         0.7384492  0.4771727
##    0   FALSE      2         0.7384492  0.4771727
##    0   FALSE      4         0.7384492  0.4771727
##    0   FALSE      6         0.7384492  0.4771727
##    0    TRUE      0               NaN        NaN
##    0    TRUE      2         0.7208540  0.4430436
##    0    TRUE      4         0.7235337  0.4492391
##    0    TRUE      6         0.7208126  0.4450891
##    1   FALSE      0         0.7384492  0.4771727
##    1   FALSE      2         0.7384492  0.4771727
##    1   FALSE      4         0.7384492  0.4771727
##    1   FALSE      6         0.7384492  0.4771727
##    1    TRUE      0               NaN        NaN
##    1    TRUE      2         0.7208540  0.4430436
##    1    TRUE      4         0.7235337  0.4492391
##    1    TRUE      6         0.7208126  0.4450891
##    2   FALSE      0         0.7384492  0.4771727
##    2   FALSE      2         0.7384492  0.4771727
##    2   FALSE      4         0.7384492  0.4771727
##    2   FALSE      6         0.7384492  0.4771727
##    2    TRUE      0               NaN        NaN
##    2    TRUE      2         0.7208540  0.4430436
##    2    TRUE      4         0.7235337  0.4492391
##    2    TRUE      6         0.7208126  0.4450891
##    3   FALSE      0         0.7384492  0.4771727
##    3   FALSE      2         0.7384492  0.4771727
##    3   FALSE      4         0.7384492  0.4771727
##    3   FALSE      6         0.7384492  0.4771727
##    3    TRUE      0               NaN        NaN
##    3    TRUE      2         0.7208540  0.4430436
##    3    TRUE      4         0.7235337  0.4492391
##    3    TRUE      6         0.7208126  0.4450891
##    4   FALSE      0         0.7384492  0.4771727
##    4   FALSE      2         0.7384492  0.4771727
##    4   FALSE      4         0.7384492  0.4771727
##    4   FALSE      6         0.7384492  0.4771727
##    4    TRUE      0               NaN        NaN
##    4    TRUE      2         0.7208540  0.4430436
##    4    TRUE      4         0.7235337  0.4492391
##    4    TRUE      6         0.7208126  0.4450891
##    5   FALSE      0         0.7384492  0.4771727
##    5   FALSE      2         0.7384492  0.4771727
##    5   FALSE      4         0.7384492  0.4771727
##    5   FALSE      6         0.7384492  0.4771727
##    5    TRUE      0               NaN        NaN
##    5    TRUE      2         0.7208540  0.4430436
##    5    TRUE      4         0.7235337  0.4492391
##    5    TRUE      6         0.7208126  0.4450891
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE and adjust
##  = 0.
```

Elastic Net Regression

```r
(eln_model <- train(x=train_x,
                    y=train_y,
                    method='glmnet',
                    trControl=cv_10,
                    tuneLength=5,
                    verbose=FALSE))
```

```
## glmnet
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1327, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.100  0.0002448348  0.7926577  0.5845333
##   0.100  0.0011364227  0.7919774  0.5831368
##   0.100  0.0052748072  0.7960270  0.5912759
##   0.100  0.0244834872  0.8021403  0.6031653
##   0.100  0.1136422858  0.7879416  0.5746592
##   0.325  0.0002448348  0.7919774  0.5831368
##   0.325  0.0011364227  0.7926577  0.5844763
##   0.325  0.0052748072  0.7953605  0.5898236
##   0.325  0.0244834872  0.7980861  0.5949178
##   0.325  0.1136422858  0.7737110  0.5461195
##   0.550  0.0002448348  0.7906261  0.5805068
##   0.550  0.0011364227  0.7933288  0.5858498
##   0.550  0.0052748072  0.7980770  0.5952489
##   0.550  0.0244834872  0.7994420  0.5975438
##   0.550  0.1136422858  0.7770895  0.5525117
##   0.775  0.0002448348  0.7906261  0.5805068
##   0.775  0.0011364227  0.7926532  0.5845404
##   0.775  0.0052748072  0.7980724  0.5950814
##   0.775  0.0244834872  0.7960407  0.5905000
##   0.775  0.1136422858  0.7736925  0.5445815
##   1.000  0.0002448348  0.7906261  0.5805068
##   1.000  0.0011364227  0.7940137  0.5872180
##   1.000  0.0052748072  0.7973875  0.5936796
##   1.000  0.0244834872  0.7960589  0.5901483
##   1.000  0.1136422858  0.7737199  0.5444914
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.02448349.
```

Decision Tree (CART)

```r
(cart_model <- train(x=train_x,
                     y=train_y,
                     method='rpart',
                     trControl=cv_10,
```

```r
                      tuneLength=5))
```

```
## CART
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1329, 1328, 1329, 1329, 1329, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.007132668  0.7873713  0.5726199
##   0.008559201  0.7859971  0.5691397
##   0.012838802  0.7717574  0.5400804
##   0.023537803  0.7629456  0.5188840
##   0.499286733  0.6027828  0.1709656
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.007132668.
```

Bagging

```r
(bag_model <- train(x=train_x,
                    y=train_y,
                    method='treebag',
                    trControl=cv_10))
```

```
## Bagged CART
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1329, 1329, 1327, 1328, 1329, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7879599  0.5732403
```

Random Forest

```r
rf_search_grid <- expand.grid(splitrule=c('gini'),
                              min.node.size=c(10, 20, 30, 40),
                              mtry=c(10, 20, 30, 40))
(rf_model <- train(x=train_x,
                   y=train_y,
                   method='ranger',
                   trControl=cv_10,
                   tuneGrid=rf_search_grid))
```

```
## Random Forest
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1328, 1328, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   min.node.size  mtry  Accuracy   Kappa
##   10             10    0.7994651  0.5954100
##   10             20    0.7988169  0.5948408
##   10             30    0.7988122  0.5947106
##   10             40    0.7995247  0.5958321
##   20             10    0.8008210  0.5985447
##   20             20    0.8001499  0.5971961
##   20             30    0.7961141  0.5890714
##   20             40    0.7954475  0.5876142
##   30             10    0.7974517  0.5918317
##   30             20    0.8015470  0.6000275
##   30             30    0.7988169  0.5943078
##   30             40    0.7947719  0.5861357
##   40             10    0.7947169  0.5863845
##   40             20    0.7995109  0.5959350
##   40             30    0.7947719  0.5860970
##   40             40    0.7968172  0.5901913
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 20, splitrule = gini
##  and min.node.size = 30.
```

Boosting

```
gbm_search_grid <- expand.grid(shrinkage=c(0.1, 0.5, 1),
                               n.minobsinnode=c(10, 20, 30, 40),
                               n.trees=c(50, 100, 200),
                               interaction.depth=c(3, 5, 10, 15))

(gbm_model <- train(x=train_x,
                   y=train_y,
                   method='gbm',
                   trControl=cv_10,
                   tuneGrid=gbm_search_grid,
                   verbose=FALSE))
```

```
## Stochastic Gradient Boosting
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1328, 1328, 1329, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.1         3                 10               50      0.7953760  0.5880078
##   0.1         3                 10              100      0.7994576  0.5963253
##   0.1         3                 10              200      0.7980971  0.5938650
##   0.1         3                 20               50      0.7953898  0.5874133
##   0.1         3                 20              100      0.7981109  0.5933007
##   0.1         3                 20              200      0.7967733  0.5909176
##   0.1         3                 30               50      0.7967503  0.5904812
##   0.1         3                 30              100      0.8055387  0.6088669
##   0.1         3                 30              200      0.8014755  0.6003913
##   0.1         3                 40               50      0.7967457  0.5903266
##   0.1         3                 40              100      0.8055663  0.6088968
##   0.1         3                 40              200      0.8055525  0.6089865
##   0.1         5                 10               50      0.7973984  0.5918331
##   0.1         5                 10              100      0.7926733  0.5824676
##   0.1         5                 10              200      0.7933536  0.5840591
##   0.1         5                 20               50      0.7899476  0.5769475
##   0.1         5                 20              100      0.8001471  0.5978956
##   0.1         5                 20              200      0.8055801  0.6089899
##   0.1         5                 30               50      0.7974352  0.5923665
##   0.1         5                 30              100      0.8021695  0.6020443
##   0.1         5                 30              200      0.8055341  0.6086233
##   0.1         5                 40               50      0.8021419  0.6013873
##   0.1         5                 40              100      0.8089079  0.6160501
##   0.1         5                 40              200      0.8062236  0.6102280
##   0.1        10                 10               50      0.8069268  0.6115542
##   0.1        10                 10              100      0.8076163  0.6129682
##   0.1        10                 10              200      0.8075979  0.6127608
##   0.1        10                 20               50      0.8068717  0.6115709
##   0.1        10                 20              100      0.8034979  0.6049107
##   0.1        10                 20              200      0.7960563  0.5896217
##   0.1        10                 30               50      0.7893179  0.5759243
##   0.1        10                 30              100      0.7927009  0.5830405
##   0.1        10                 30              200      0.7947233  0.5872655
##   0.1        10                 40               50      0.8041965  0.6056068
##   0.1        10                 40              100      0.8014846  0.6004507
##   0.1        10                 40              200      0.7960838  0.5900040
##   0.1        15                 10               50      0.7906417  0.5784463
##   0.1        15                 10              100      0.8042195  0.6059143
##   0.1        15                 10              200      0.8008090  0.5993367
##   0.1        15                 20               50      0.7987590  0.5948311
##   0.1        15                 20              100      0.7892949  0.5761360
##   0.1        15                 20              200      0.7987957  0.5951065
##   0.1        15                 30               50      0.7981109  0.5932695
##   0.1        15                 30              100      0.7987911  0.5952566
##   0.1        15                 30              200      0.7967779  0.5910264
##   0.1        15                 40               50      0.8048722  0.6070091
##   0.1        15                 40              100      0.8022063  0.6019010
##   0.1        15                 40              200      0.7981568  0.5941667
##   0.5         3                 10               50      0.7683168  0.5346494
```

```
##    0.5     3     10     100    0.7527165  0.5039396
##    0.5     3     10     200    0.7581449  0.5150090
##    0.5     3     20      50    0.7791230  0.5560904
##    0.5     3     20     100    0.7852638  0.5686618
##    0.5     3     20     200    0.7832046  0.5650763
##    0.5     3     30      50    0.7872587  0.5724416
##    0.5     3     30     100    0.7723754  0.5430947
##    0.5     3     30     200    0.7703438  0.5391325
##    0.5     3     40      50    0.7838803  0.5653438
##    0.5     3     40     100    0.7737314  0.5457056
##    0.5     3     40     200    0.7723754  0.5429341
##    0.5     5     10      50    0.7709643  0.5404638
##    0.5     5     10     100    0.7696819  0.5372586
##    0.5     5     10     200    0.7764433  0.5510714
##    0.5     5     20      50    0.7730189  0.5441813
##    0.5     5     20     100    0.7682708  0.5351586
##    0.5     5     20     200    0.7689971  0.5364790
##    0.5     5     30      50    0.7784933  0.5552503
##    0.5     5     30     100    0.7669654  0.5321443
##    0.5     5     30     200    0.7771511  0.5525679
##    0.5     5     40      50    0.7750919  0.5483234
##    0.5     5     40     100    0.7669470  0.5323104
##    0.5     5     40     200    0.7771098  0.5528145
##    0.5    10     10      50    0.7615049  0.5218003
##    0.5    10     10     100    0.7778038  0.5538878
##    0.5    10     10     200    0.7839125  0.5657086
##    0.5    10     20      50    0.7825519  0.5632620
##    0.5    10     20     100    0.7717044  0.5417078
##    0.5    10     20     200    0.7825565  0.5634040
##    0.5    10     30      50    0.7784611  0.5551460
##    0.5    10     30     100    0.7798125  0.5580367
##    0.5    10     30     200    0.7804927  0.5596314
##    0.5    10     40      50    0.7750735  0.5487726
##    0.5    10     40     100    0.7778268  0.5545462
##    0.5    10     40     200    0.7859395  0.5698597
##    0.5    15     10      50    0.7751011  0.5477728
##    0.5    15     10     100    0.7717090  0.5413982
##    0.5    15     10     200    0.7832322  0.5645791
##    0.5    15     20      50    0.7839033  0.5657754
##    0.5    15     20     100    0.7818625  0.5616447
##    0.5    15     20     200    0.7927238  0.5837519
##    0.5    15     30      50    0.7757308  0.5495856
##    0.5    15     30     100    0.7743887  0.5468240
##    0.5    15     30     200    0.7825198  0.5632943
##    0.5    15     40      50    0.7717181  0.5412209
##    0.5    15     40     100    0.7595100  0.5166413
##    0.5    15     40     200    0.7730327  0.5437854
##    1.0     3     10      50    0.7412116  0.4822737
##    1.0     3     10     100    0.7425216  0.4853228
##    1.0     3     10     200    0.7195394  0.4388879
##    1.0     3     20      50    0.7324232  0.4638873
##    1.0     3     20     100    0.7404946  0.4805270
##    1.0     3     20     200    0.7324278  0.4642906
##    1.0     3     30      50    0.7445900  0.4872932
```

```
##    1.0          3              30          100      0.7439235  0.4861245
##    1.0          3              30          200      0.7466308  0.4914464
##    1.0          3              40           50      0.7548079  0.5085782
##    1.0          3              40          100      0.7344181  0.4672170
##    1.0          3              40          200      0.7391662  0.4766247
##    1.0          5              10           50      0.7195211  0.4385438
##    1.0          5              10          100      0.7053411  0.4110137
##    1.0          5              10          200      0.6422274  0.2819501
##    1.0          5              20           50      0.7398695  0.4779873
##    1.0          5              20          100      0.7385227  0.4763366
##    1.0          5              20          200      0.7100202  0.4202406
##    1.0          5              30           50      0.7174756  0.4335813
##    1.0          5              30          100      0.7317154  0.4610735
##    1.0          5              30          200      0.7466216  0.4921694
##    1.0          5              40           50      0.7418551  0.4834302
##    1.0          5              40          100      0.7228535  0.4452564
##    1.0          5              40          200      0.7236027  0.4475758
##    1.0         10              10           50      0.6761813  0.3531440
##    1.0         10              10          100      0.5832828  0.1611698
##    1.0         10              10          200      0.5832828  0.1611698
##    1.0         10              20           50      0.7188500  0.4364140
##    1.0         10              20          100      0.6476604  0.2928900
##    1.0         10              20          200      0.6537691  0.3073947
##    1.0         10              30           50      0.7059570  0.4112556
##    1.0         10              30          100      0.6634400  0.3266936
##    1.0         10              30          200      0.6084161  0.2158289
##    1.0         10              40           50      0.7236624  0.4460734
##    1.0         10              40          100      0.6903475  0.3770543
##    1.0         10              40          200      0.6143547  0.2249978
##    1.0         15              10           50      0.6422274  0.2860731
##    1.0         15              10          100      0.6314166  0.2660972
##    1.0         15              10          200      0.6314166  0.2660972
##    1.0         15              20           50      0.6863440  0.3720140
##    1.0         15              20          100      0.6035990  0.2083306
##    1.0         15              20          200      0.6035990  0.2083306
##    1.0         15              30           50      0.7012364  0.4015794
##    1.0         15              30          100      0.5806720  0.1598586
##    1.0         15              30          200      0.5840734  0.1674432
##    1.0         15              40           50      0.7303273  0.4600830
##    1.0         15              40          100      0.6666942  0.3351029
##    1.0         15              40          200      0.6450726  0.2898281
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth =
##  5, shrinkage = 0.1 and n.minobsinnode = 40.
```

Evaluate the models

3.

Logistic Regression

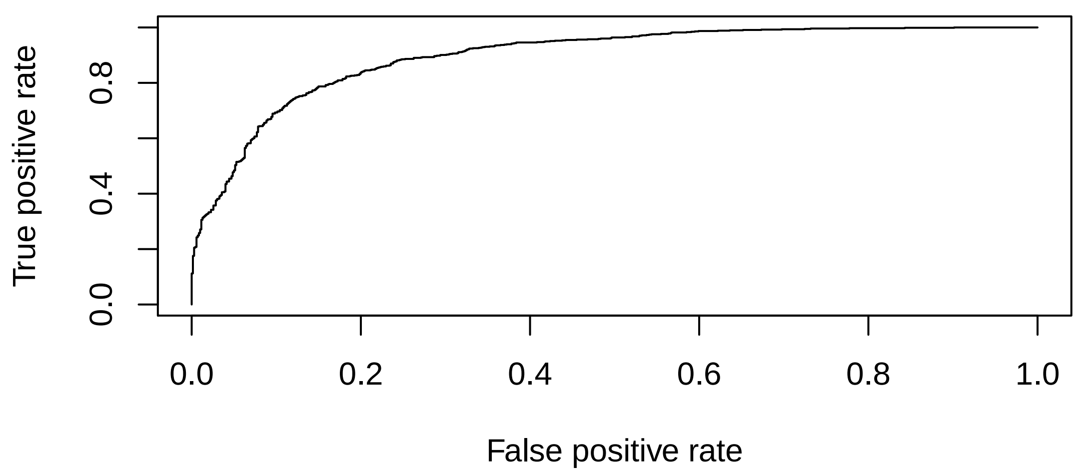
```r
# Cross-validated error rate
1 - max(log_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.2018906
```

```r
# ROC/AUC
fit <- predict(log_model$finalModel,
               newdata=df_train)
pred <- prediction(fit, train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.8965404
```

```r
# ROC curve
plot(perf)
```



Naive Bayes

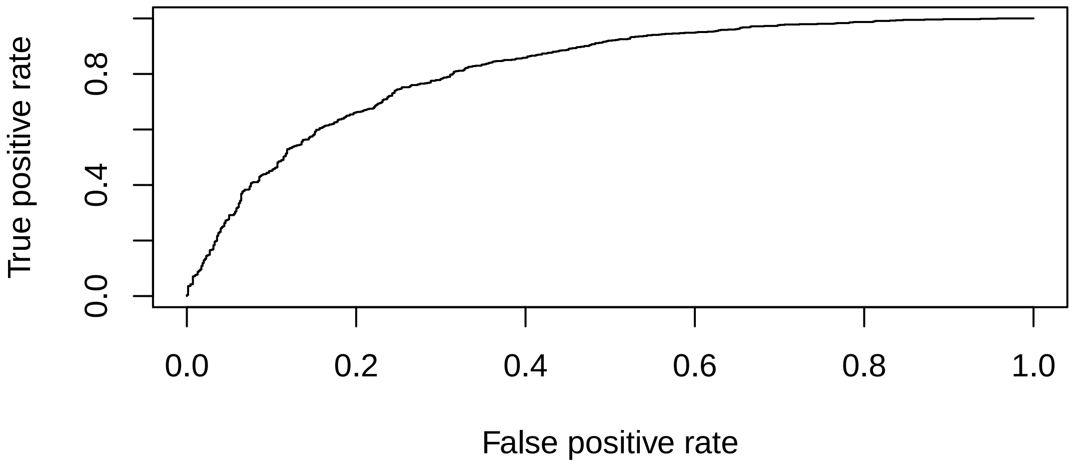
```r
# Cross-validated error rate
1 - max(nb_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.2615508
```

```r
# ROC/AUC
fit <- as.numeric(stats::predict(nb_model$finalModel,
                newdata=train_x)$posterior[,2])
pred <- ROCR::prediction(fit, as.numeric(train_y))
perf <- ROCR::performance(pred,'tpr', 'fpr')
# AUC
ROCR::performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.8164134
```

```r
# ROC plot
plot(perf)
```



Elastic Net Regression

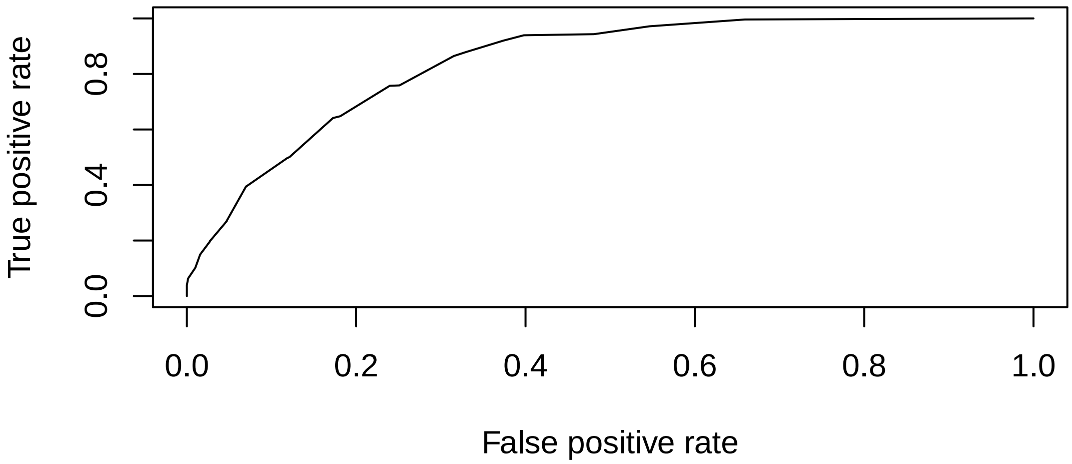
```
# Cross-validated error rate
1 - max(eln_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.1978597
```

```
# ROC/AUC
fit <- predict(eln_model$finalModel,
               newx=train_x)
pred <- prediction(fit[,2], train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.8420818
```

```
# ROC curve
plot(perf)
```



Decision Tree (CART)

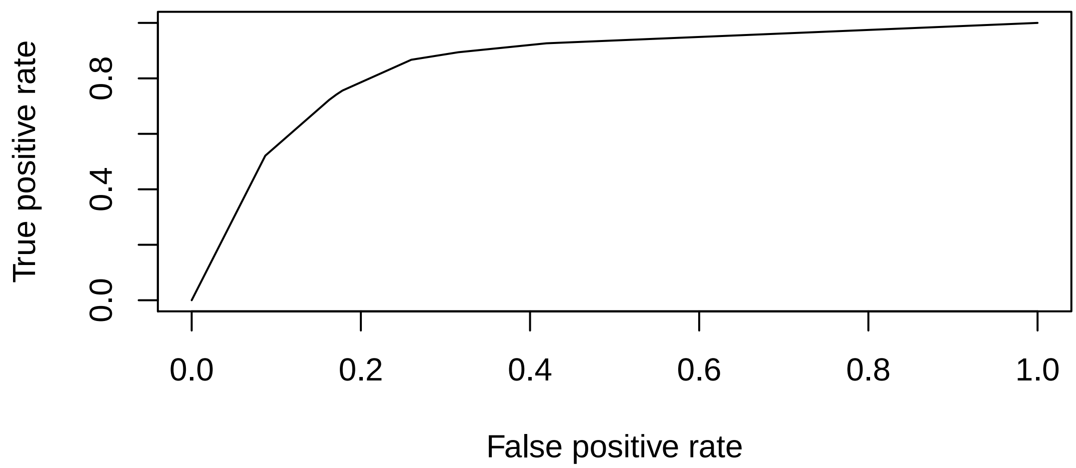
```
# Cross-validated error rate
1 - max(cart_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.2126287
```

```
# ROC/AUC
fit <- predict(cart_model$finalModel,
               newdata=df_train)
pred <- prediction(fit[,2], train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.8503484
```

```
# ROC curve
plot(perf)
```



Bagging

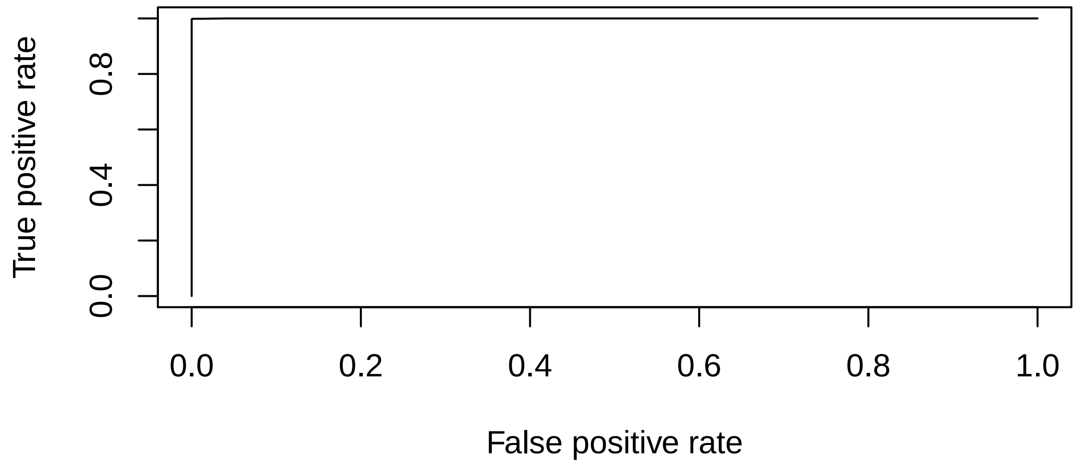
```
# Cross-validated error rate
1 - max(bag_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.2120401
```

```
# ROC/AUC
fit <- predict(bag_model$finalModel,
               newdata=df_train, type='prob')
pred <- prediction(fit[,2], train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.9999641
```

```
# ROC curve
plot(perf)
```



Random Forest

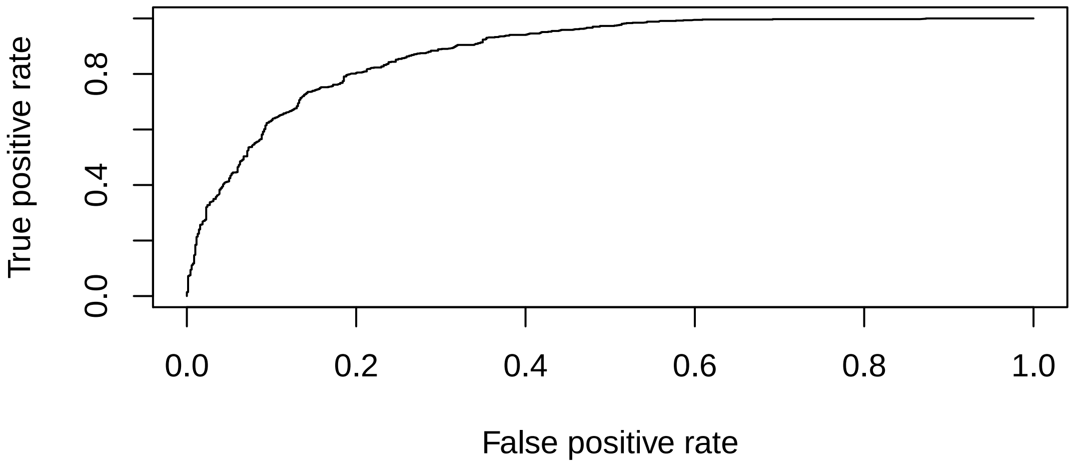
```
# Cross-validated error rate
1 - max(rf_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.198453
```

```
# ROC/AUC
rf_model_fin <- randomForest::randomForest(colrac~.,
                                           data=df_train,
                                           mtry=rf_model$bestTune$mtry,
                                           nodesize=rf_model$bestTune$min.node.size)
fit <- predict(rf_model_fin,
               data=df_train, type='prob')
pred <- prediction(fit[,2], train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.8833951
```

```
# ROC curve
plot(perf)
```



Boosting

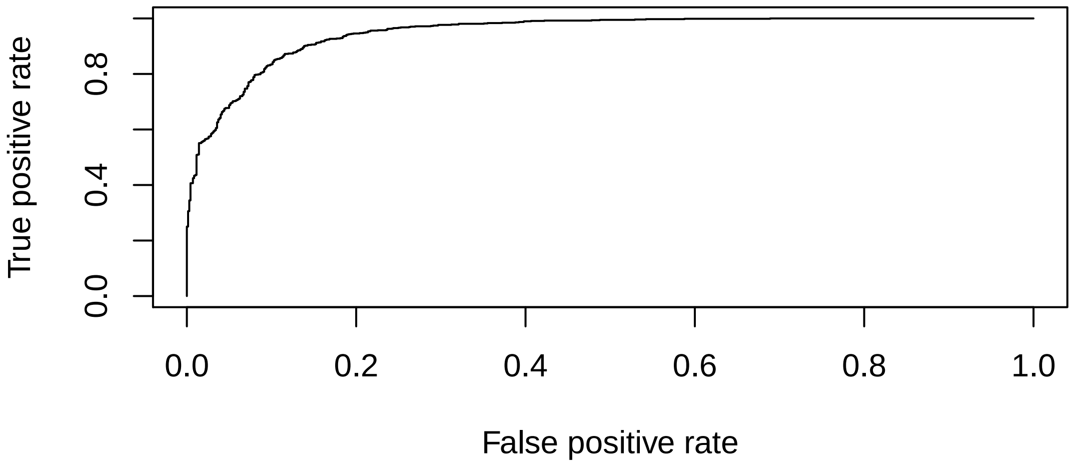
```
# Cross-validated error rate
1- max(gbm_model$results$Accuracy, na.rm=TRUE)
```

```
## [1] 0.1910921
```

```
# ROC/AUC
fit <- predict(gbm_model,
               newdata=df_train,
               type='prob')
pred <- prediction(fit[,2], train_y)
perf <- performance(pred, 'tpr', 'fpr')
# AUC
performance(pred, measure='auc')@y.values[[1]]
```

```
## [1] 0.9491952
```

```
# ROC curve
plot(perf)
```



4.

According to the results above, the Bagging has the highest AUC, which is almost 1.0, and Boosting has second highest AUC, which is around 0.95. Those models also have low error rate; both are around 0.21 and 0.19. On the other hand, logistic regression's error rate is around 0.2 and its AUC is around 0.90. Therefore, although Bagging has almost 1 AUC, its accuracy is lower than logistic regression. From this results, I prefer to use Boosting because it has both high AUC and low error rate.

Evaluate the best model

5.

Evaluate with test data set.

Logistic Regression

```
fit <- predict(log_model$finalModel,
               newdata=df_test)
pred <- prediction(fit, test_y)
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##    accuracy  cutoff.234         auc
##  0.79513185 -0.02362787  0.86191658
```

Naive Bayes

```r
fit <- as.numeric(stats::predict(nb_model$finalModel,
                newdata=test_x)$posterior[,2])
pred <- ROCR::prediction(fit, as.numeric(test_y))
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##  accuracy    cutoff       auc
## 0.7444219 0.1461593 0.8060741
```

Elastic Net Regression

```r
fit <- predict(eln_model$finalModel,
               newx=test_x)
pred <- prediction(fit[,2], test_y)
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##   accuracy cutoff.478        auc
##  0.7849899  0.1015159  0.8369580
```

Decision Tree (CART)

```r
fit <- predict(cart_model$finalModel,
               newdata=df_test)
pred <- prediction(fit[,2], test_y)
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##   accuracy cutoff.487        auc
##  0.7951318  0.6013986  0.8306273
```

Bagging

```r
fit <- predict(bag_model$finalModel,
               newdata=df_test, type='prob')
pred <- prediction(fit[,2], test_y)
acc.perf <- performance(pred, measure='acc')
```

```
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##  accuracy    cutoff       auc
## 0.8093306 0.4800000 0.8687024
```

Random Forest

```
fit <- predict(rf_model_fin,
               newdata=df_test, type='prob')
pred <- prediction(fit[,2], test_y)
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##   accuracy cutoff.230        auc
##  0.7971602  0.5800000  0.8641344
```

Boosting

```
fit <- predict(gbm_model,
               newdata=df_test, type='prob')
pred <- prediction(fit[,2], test_y)
acc.perf <- performance(pred, measure='acc')
auc <- performance(pred, measure='auc')@y.values[[1]]

ind <- which.max(slot(acc.perf, "y.values")[[1]])
acc <- slot(acc.perf, "y.values")[[1]][ind]
cutoff <- slot(acc.perf, "x.values")[[1]][ind]

print(c(accuracy=acc, cutoff=cutoff, auc=auc))
```

```
##  accuracy    cutoff       auc
## 0.8093306 0.4805752 0.8715823
```

In terms of the accuracy, the results from the test dataset is not so different from what we got from the train dataset. This implies that we did avoid over-fitting by using cross-validation. On the other hand, when considering the AUC, Bagging and Boosting have lower AUC than what we had with train dataset. This could be resulted from overfitting. The AUC of Bagging and Boosting are now close to that of logistic regression. Therefore, in terms of accuracy and AUC, those models are indifferent. To choose the best model, therefore, we have to consider other criteria such as interpretability. In this case, Logistic Regression is the best model from interpretability.