# Li_Hengle_hw5

Hengle Li

2/27/2020

## 1. Conceptual

- The Gini index is the best in growing a decision tree in a setting with two classes. The algorithm of the decision tree aims to find the feature and splitting value that leads to the largest information gain. In a two-class case, say 4 balls of either red color or blue color, the information gain is: `IG = I(D0) - (N1/N0)*I(D1) - (N2/N0)*I(D2)`, where I can be entropy, Gini index, or classification error, and D0, D1, D2 are the dataset of the parent and children nodes. The information gain for classification error is the same whether one slipts the 4 balls by 2:2 or 1:3, while entropy and Gini index will point to 1:3. Between the latter two, repeating the same experiment would show that Gini index can produce a higher information gain than entropy.
- Classification error rate is the best in pruning a decision tree. The goal of pruning is to simplify decision trees that overfit the data. In this respect, classification error rate is more sensitive to overfitting than the other two measures, which makes it more preferable.

```r
library(tidyverse)
library(tidymodels)
library(rcfss)
library(margins)
library(rsample)
library(caret)
library(glmnet)
library(yardstick)
library(tree)
library(randomForest)
library(ranger)
library(pROC)
library(iml)

set.seed(6758)
```

```r
gss_test <- read_csv("data/gss_test.csv")
gss_train <- read_csv("data/gss_train.csv")
```

```r
#spliting the training set for model fitting
spliting <- initial_split(gss_train)
gss_train_train <- training(spliting)
gss_train_test <- testing(spliting)

#naive Bayesian in train() requires categorical values
gss_train_cat <- gss_train_train %>%
  mutate(colrac_cat = ifelse(colrac == 1, "Allowed", "Not allowed"))
```

```r
gss_train_cat2 <- gss_train_test %>%
  mutate(colrac_cat = ifelse(colrac == 1, "Allowed", "Not allowed"))

#10-fold CV
gss_train_cv <- gss_train_train %>% vfold_cv(v = 10)
```

## 2. Estimate models

```r
#set up X and Y for models
X <- gss_train_train %>%
  dplyr::select(-colrac)

Y <- gss_train_train$colrac

#categorical values for naive Bayesian
Y_cat <- gss_train_cat$colrac_cat

#for elastic net
X_cv <- model.matrix(colrac ~ ., data = gss_train_train)[, -1]
X_cv_test <- model.matrix(colrac ~ ., data = gss_train_test)[, -1]
```

### Logistic regression

```r
#define control method to k-fold CV
cv_10 <- trainControl(method = "cv", number = 10)

#construct model
glm_gss <- train(
  x = X,
  y = Y,
  method = "glm",
  trControl = cv_10,
  family = "binomial"
)
```

```r
#make predictions and present results
glm_result <- tibble(truth = gss_train_test$colrac,
                     pred = round(predict(glm_gss, newdata = gss_train_test))) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

glm_result$err_rate
```

```
## [1] 20.59621
```

### Naive Bayesian

```r
#construct model
nb_gss <- train(
  x = X,
  y = Y_cat,
  method = "nb",
  trControl = cv_10
)
```

```r
#accuracy through confusionMatrix()
confusionMatrix(nb_gss)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##              Reference
## Prediction    Allowed Not allowed
##   Allowed        39.3        11.9
##   Not allowed    14.5        34.3
##
##  Accuracy (average) : 0.7362
```

```r
#error rate by direct calculations
nb_result <- tibble(truth = gss_train_cat2$colrac_cat,
                    pred = predict(nb_gss, newdata = gss_train_test)) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`),
         accuracy = 100* `TRUE` / (`FALSE` + `TRUE`))

#accuracy
nb_result$accuracy
```

```
## [1] 72.35772
```

```r
#error rate
nb_result$err_rate
```

```
## [1] 27.64228
```

### Elastic net

```r
#randomize which fold among the 10 to use for training
folds <- sample(1:10, size = length(Y), replace = TRUE)

#create a grid for entries of lambda and MSE
#using lambda.1se here as in the previous two models
```

```r
tuning <- tibble(alpha = seq(0, 1, by = .1),
                 mse_1se = NA,
                 lambda_1se = NA
                 )

#fill in the grid with values
#for each alpha, there is a corresponding lambda
for(i in seq_along(tuning$alpha)){
  filling <- cv.glmnet(x = X_cv,
                   y = Y,
                   alpha = tuning$alpha[i],
                   foldid = folds)
  tuning$mse_1se[i] <- filling$cvm[filling$lambda == filling$lambda.1se]
  tuning$lambda_1se[i] <- filling$lambda.1se
}

#create another grid for the calculated lambda and possible alpha
testing <- expand.grid(alpha = seq(0, 1, by = 0.1),
                       lambda_1se = tuning$lambda_1se)
```

```r
#a grid for all models and their testing MSE
models_enet <- tibble(alpha = testing$alpha,
                      lambda_1se = testing$lambda_1se,
                      err_rate = NA)

#use a function to make predictions
very_fancy <- function(al, bi){
  #reproduce the model with corresponding alpha
  dumb <- glmnet(
    x = X_cv,
    y = Y,
    alpha = al
    )
  mid_filler <- tibble(truth = gss_train_train$colrac,
                       pred = round(predict(dumb,
                                  s = bi,
                                  newx = X_cv))) %>%
    #calculate error rates
    count(truth == pred) %>%
    spread("truth == pred", n) %>%
    mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))
  mid_filler$err_rate
}
```

```r
#execute the calculations
for(i in seq_along(models_enet$alpha)) {
  models_enet$err_rate[i] <- very_fancy(models_enet$alpha[i],
                                  models_enet$lambda_1se[i])
}

#present the results
models_enet
```

```
## # A tibble: 121 x 3
##    alpha lambda_1se err_rate
##    <dbl>      <dbl>    <dbl>
##  1   0        0.357     19.2
##  2   0.1      0.357     22.7
##  3   0.2      0.357     22.5
##  4   0.3      0.357     22.5
##  5   0.4      0.357     21.5
##  6   0.5      0.357     21.8
##  7   0.6      0.357     26.4
##  8   0.7      0.357     46.3
##  9   0.8      0.357     46.3
## 10   0.9      0.357     46.3
## # ... with 111 more rows
```

```r
#find the combination where MSE is smallest
elas_meters <- models_enet[which.min(models_enet$err_rate),]
elas_meters
```

```
## # A tibble: 1 x 3
##   alpha lambda_1se err_rate
##   <dbl>      <dbl>    <dbl>
## 1     0     0.0391     17.2
```

```r
#tune glmnet based on the results
elas_gss <- glmnet(
  x = X_cv,
  y = Y,
  alpha = elas_meters$alpha
  )

elas_result <- tibble(truth = gss_train_test$colrac,
                      pred = round(predict(elas_gss,
                                           s = elas_meters$lambda_1se,
                                           newx = X_cv_test))) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

elas_result$err_rate
```

```
## [1] 20.59621
```

### Decision tree

```r
rac_tree <- vfold_cv(gss_train_train, v = 10) %>%
  mutate(tree = map(splits, ~ tree(colrac ~ .,
                                   data = analysis(.x),
                                   control = tree.control(nobs = nrow(gss_train_train),
                                                          mindev = 0))))
```

```r
# generate 10-fold CV trees
colrac_cv <- vfold_cv(gss_train_train, v = 10) %>%
  #a full tree for each split
  mutate(tree = map(splits, ~ tree(colrac ~ ., data = analysis(.x),
                                    control = tree.control(nobs = nrow(gss_train_train),
                                                           mindev = 0))))

# calculate each possible prune result for each fold
colrac_prune <- expand.grid(rac_tree$id, 2:50) %>%
  as_tibble() %>%
  mutate(Var2 = as.numeric(Var2)) %>%
  rename(id = Var1,
         k = Var2) %>%
  left_join(colrac_cv) %>%
  mutate(prune = map2(tree, k, ~ prune.tree(.x, best = .y)),
         estimate = map2(prune, splits, ~ predict(.x, newdata = assessment(.y))),
         truth = map(splits, ~ assessment(.x)$colrac)) %>%
  unnest(estimate, truth) %>%
  #round estimates for comparison
  mutate(estimate = round(estimate)) %>%
  group_by(k) %>%
  count(truth == estimate) %>%
  spread("truth == estimate", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))
```

```r
#locate the tree with the smallest error rate
tree_meters <- colrac_prune[which.min(colrac_prune$err_rate),]

tree_meters
```

```
## # A tibble: 1 x 4
## # Groups:   k [1]
##       k `FALSE` `TRUE` err_rate
##   <dbl>   <int>  <int>    <dbl>
## 1    12     241    866     21.8
```

```r
#name the tree
tree_gss <- tree(colrac ~ .,
                 data = gss_train_train,
                 control = tree.control(nobs = nrow(gss_train_train),
                                        mindev = 0)) %>%
  prune.tree(best = tree_meters$k)
```

```r
#make complete predictions with the tree
tree_result <- tibble(truth = gss_train_test$colrac,
                      pred = predict(tree_gss, newdata = gss_train_test)) %>%
  mutate(pred = round(pred)) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

#present the error rate
tree_result$err_rate
```

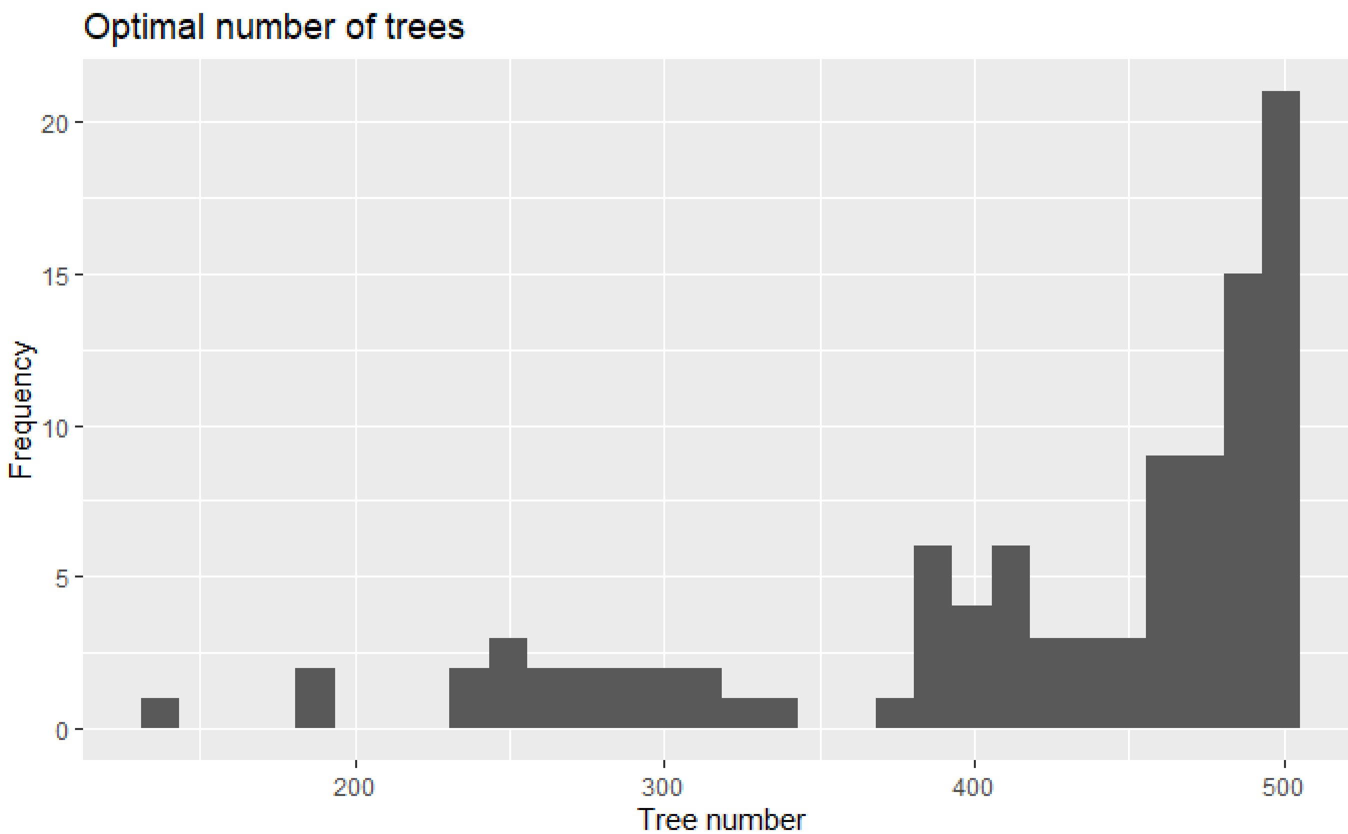
```
## [1] 27.64228
```

### Number of trees

The following codes can produce the graph below, but it takes a long time to run. Note that here comparisons are about MSE rather than error rates. Even though they are not the same thing, both can reflect the accuracy of a model, since predictions are numbers between 0 and 1. The codes here use MSE to save time. The point is, out-of-bag MSE reaches minimum around 500, which is the reason why we will be setting tree numbers to be 500 in the following models.

```
treecount <- vector(mode = "numeric", length = 100)

for(i in seq_along(treecount)) {
  tree_num <- randomForest(
    formula = colrac ~ .,
    data = gss_train)

  treecount[i] <- which.min(tree_num$mse)
}

many_trees <- tibble(number = treecount)

many_trees %>%
  ggplot(aes(x = number)) +
  geom_histogram() +
  labs(title = "Optimal number of trees",
       x = "Tree number",
       y = "Frequency")
```

```
knitr::include_graphics('manytrees.png')
```




**Bagging**

```
#create a grid for mtry, min.node.size, sample.fraction
bag_grid <- expand.grid(
  node_size = seq(1, 9, by = 1),
  sample_size = c(0.2, 0.4, 0.6, 0.8),
  err_rate = NA
)

#a list to identify which split from the CV to use
id <- sample(1:10, size = 36, replace = TRUE)

#join the grid with fold id's
bag_grid <- bind_cols(bag_grid, id = id)

#10-fold split data and assign ids
split_grid <- tibble(gss_train_cv$splits, 1:10) %>%
  rename("splits" = `gss_train_cv$splits`,
         "id" = `1:10`)

#set up the grid
bag_grid <- left_join(bag_grid, split_grid)
```

```r
for(i in 1:nrow(bag_grid)) {
  # train bagging model
  model <- randomForest(
    formula = colrac ~ .,
    data = analysis(bag_grid$splits[[i]]),
    num.trees = 500,
    mtry = ncol(analysis(bag_grid$splits[[i]])) - 1,
    replace = FALSE,
    samplesize = bag_grid$sample_size[i] * nrow(analysis(bag_grid$splits[[i]])),
    min.node.size = bag_grid$node_size[i]
  )
  # extract testing data
  truth_ext <- assessment(bag_grid$splits[[i]])
  # make predictions and calulate error rates
  model_result <- tibble(truth = truth_ext$colrac,
                         pred = round(predict(model, newdata = truth_ext))) %>%
    count(truth == pred) %>%
    spread("truth == pred", n) %>%
    mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))
  #insert error rates
  bag_grid$err_rate[i] <- model_result$err_rate
}
```

```r
# find the parameters with the lowest error rate
bag_meters <- bag_grid[which.min(bag_grid$err_rate),] %>%
  dplyr::select(-splits, -id)

bag_meters
```

```
##    node_size sample_size err_rate
## 10         1         0.4 10.90909
```

```r
# fit model
bag_gss <- randomForest(
    formula = colrac ~ .,
    data = gss_train_train,
    num.trees = 500,
    mtry = ncol(gss_train_train) - 1,
    replace = FALSE,
    samplesize = bag_meters$sample_size * nrow(gss_train_train),
    min.node.size = bag_meters$node_size
    )
```

```r
#using predict() on bag_gss will take incredibly long
#fortunately bag_gss already has predictions
bag_result <- tibble(truth = gss_train_test$colrac,
                     pred = round(predict(bag_gss, newdata = gss_train_test))) %>%
  mutate(pred = round(pred)) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))
```

```r
#present the error rate
bag_result$err_rate
```

```
## [1] 24.11924
```

### Random forest

```r
#create a grid for mtry, min.node.size, sample.fraction
rf_grid <- expand.grid(
  mtry = seq(20, 30, by = 2),
  node_size = seq(1, 9, by = 1),
  sample_size = c(0.2, 0.4, 0.6, 0.8),
  err_rate = NA
)

#a list to identify which split from the CV to use
id <- sample(1:10, size = nrow(rf_grid), replace = TRUE)

#join the grid with fold id's
rf_grid <- bind_cols(rf_grid, id = id)

#set up the grid
rf_grid <- left_join(rf_grid, split_grid)
```

```r
for(i in 1:nrow(rf_grid)) {
  # train bagging model
  model <- ranger(
    formula = colrac ~ .,
    data = analysis(rf_grid$splits[[i]]),
    num.trees = 500,
    mtry = rf_grid$mtry[i],
    replace = FALSE,
    sample.fraction = rf_grid$sample_size[[i]],
    min.node.size = rf_grid$node_size[i]
  )
  # extract testing data
  truth_ext <- assessment(rf_grid$splits[[i]])
  # make predictions, note that here in the predict function
  #new data is "data", not "newdata"
  tatakai <- predict(model, data = truth_ext)
  # calculate error rates
  model_result <- tibble(truth = truth_ext$colrac,
                         pred = round(tatakai$predictions)) %>%
    count(truth == pred) %>%
    spread("truth == pred", n) %>%
    mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))
  #insert error rates
  rf_grid$err_rate[i] <- model_result$err_rate
}
```

```r
# find the parameters with the lowest error rate
rf_meters <- rf_grid[which.min(rf_grid$err_rate),] %>%
  dplyr::select(-splits, -id)

rf_meters
```

```
##     mtry node_size sample_size err_rate
## 187   20         5         0.8 9.090909
```

```r
# fit model
rf_gss <- ranger(
    formula = colrac ~ .,
    data = gss_train_train,
    num.trees = 500,
    mtry = rf_meters$mtry,
    replace = FALSE,
    sample.fraction = rf_meters$sample_size,
    min.node.size = rf_meters$node_size
  )
```

```r
#make predictions with the tuned model
rf_ttk <- predict(rf_gss, data = gss_train_test)
#insert values and calculate error rates
rf_result <- tibble(truth = gss_train_test$colrac,
                    pred = round(rf_ttk$predictions)) %>%
  mutate(pred = round(pred)) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

#present the error rate
rf_result$err_rate
```

```
## [1] 21.40921
```

### Boosting

```r
boost_gss <- train(colrac ~ .,
                   data = gss_train_train,
                   method = "gbm",
                   trControl = cv_10,
                   verbose = 0)
```

```r
#make predictions and present results
boost_result <- tibble(truth = gss_train_test$colrac,
                    pred = round(predict(boost_gss, newdata = gss_train_test))) %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

boost_result$err_rate
```

```
## [1] 22.22222
```

### Model selection

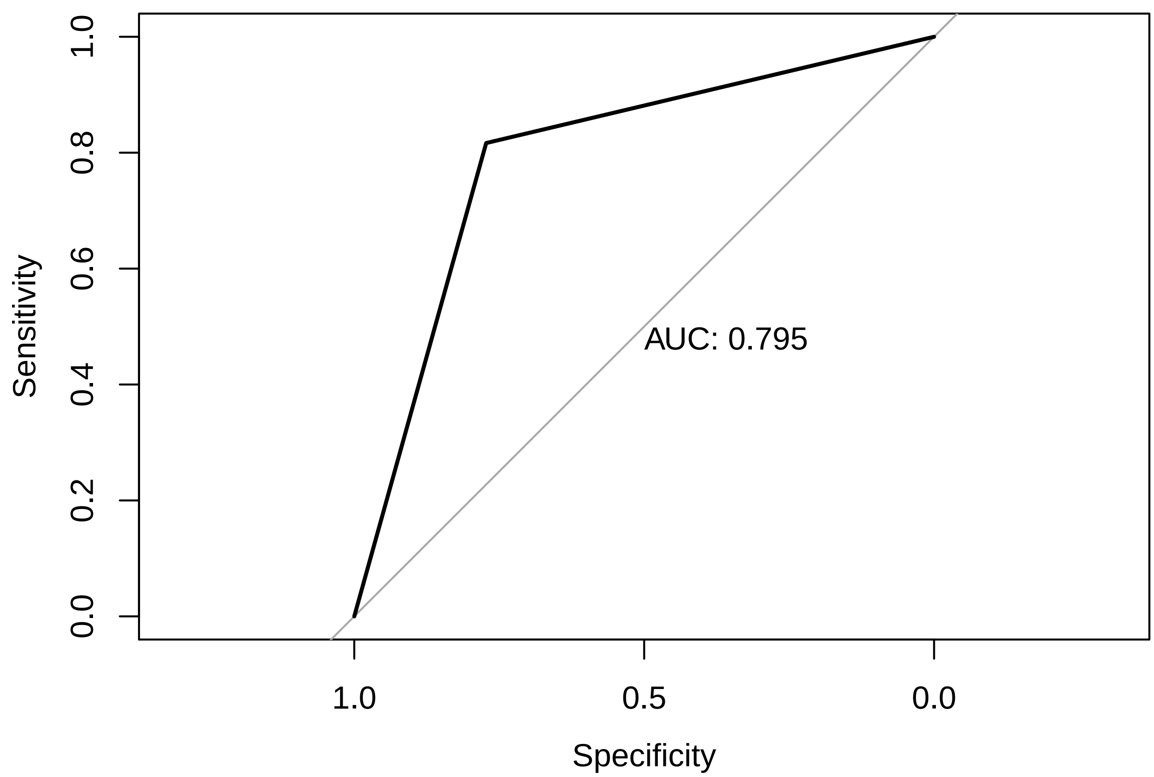
```r
error_rates <- tibble("Logistic regression" = glm_result$err_rate,
                      "Naive Bayesian" = nb_result$err_rate,
                      "Elastic net" = elas_result$err_rate,
                      "Decision tree" = tree_result$err_rate,
                      "Bagging" = bag_result$err_rate,
                      "Random forest" = rf_result$err_rate,
                      "Boosting" = boost_result$err_rate
                      )

error_rates <- data.frame(t(error_rates))
error_rates
```

```
##                     t.error_rates.
## Logistic regression       20.59621
## Naive Bayesian            27.64228
## Elastic net               20.59621
## Decision tree             27.64228
## Bagging                   24.11924
## Random forest             21.40921
## Boosting                  22.22222
```

```r
#glm
glm_pred <- tibble(truth = gss_train_test$colrac,
                    pred = round(predict(glm_gss, newdata = gss_train_test)))

#nb
nb_pred <- tibble(truth = gss_train_cat2$colrac_cat,
                    pred = predict(nb_gss, newdata = gss_train_test))

#elastic net
elas_pred <- tibble(truth = gss_train_test$colrac,
                      pred = round(predict(elas_gss,
                                           s = elas_meters$lambda_1se,
                                           newx = X_cv_test)))

#decision tree
tree_pred <- tibble(truth = gss_train_test$colrac,
                      pred = predict(tree_gss, newdata = gss_train_test))

#bagging
bag_pred <- tibble(truth = gss_train_test$colrac,
                      pred = round(predict(bag_gss, newdata = gss_train_test)))

#random forest
rf_pred <- tibble(truth = gss_train_test$colrac,
                      pred = round(rf_ttk$predictions))
```

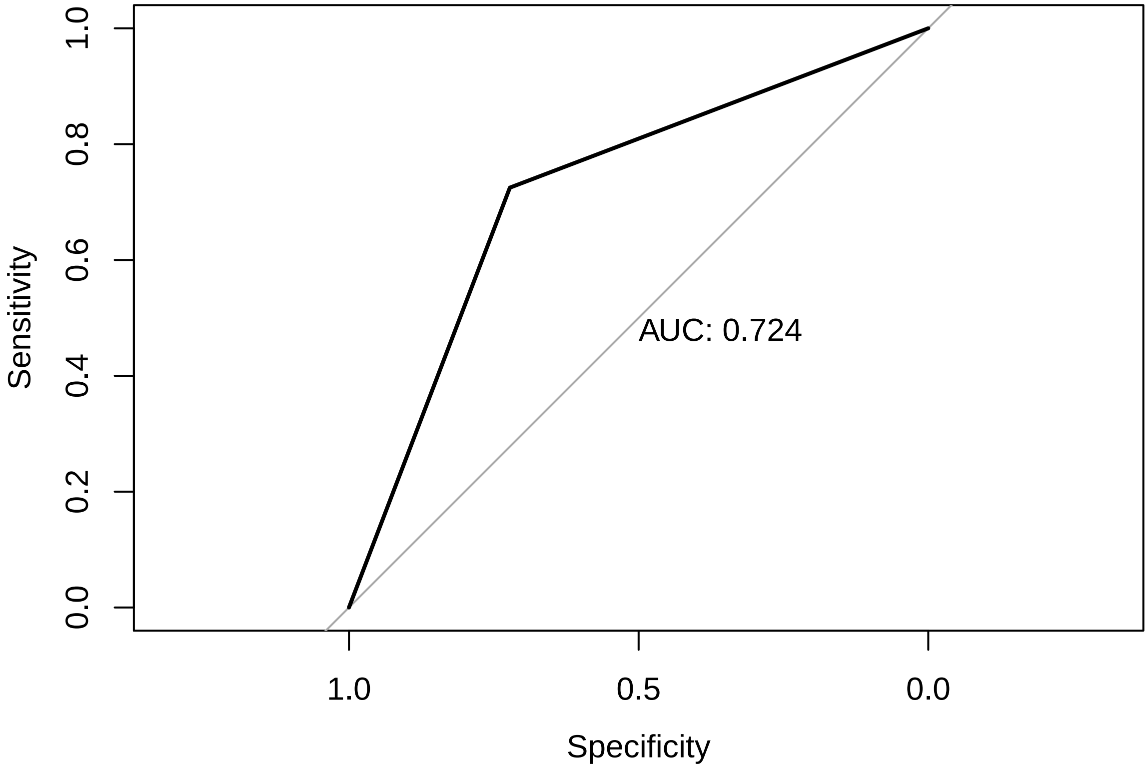
```r
#boosting
boost_pred <- tibble(truth = gss_train_test$colrac,
                     pred = round(predict(boost_gss, newdata = gss_train_test)))
```

```r
#glm
roc(response = glm_pred$truth,
               predictor =  as.numeric(glm_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```



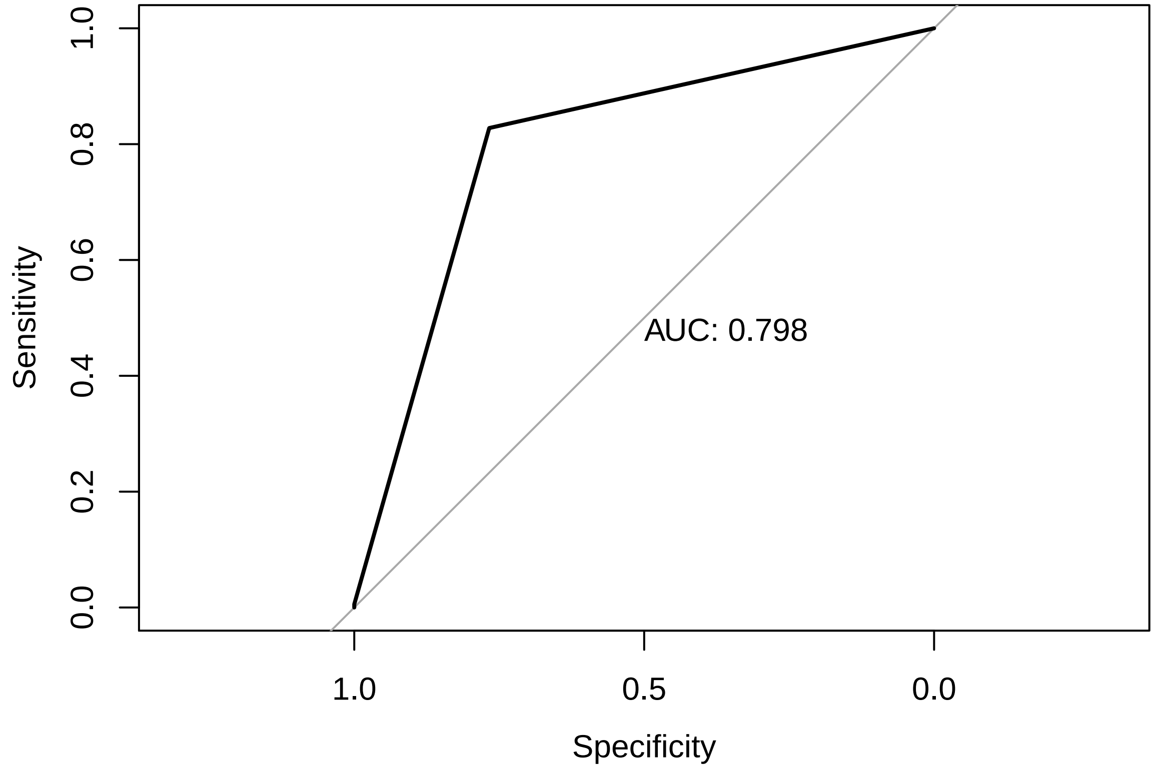

```
##
## Call:
## roc.default(response = glm_pred$truth, predictor = as.numeric(glm_pred$pred),     plot = TRUE, print
##
## Data: as.numeric(glm_pred$pred) in 189 controls (glm_pred$truth 0) < 180 cases (glm_pred$truth 1).
## Area under the curve: 0.7946
```

```r
#nb
roc(response = nb_pred$truth,
               predictor =  as.numeric(nb_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```



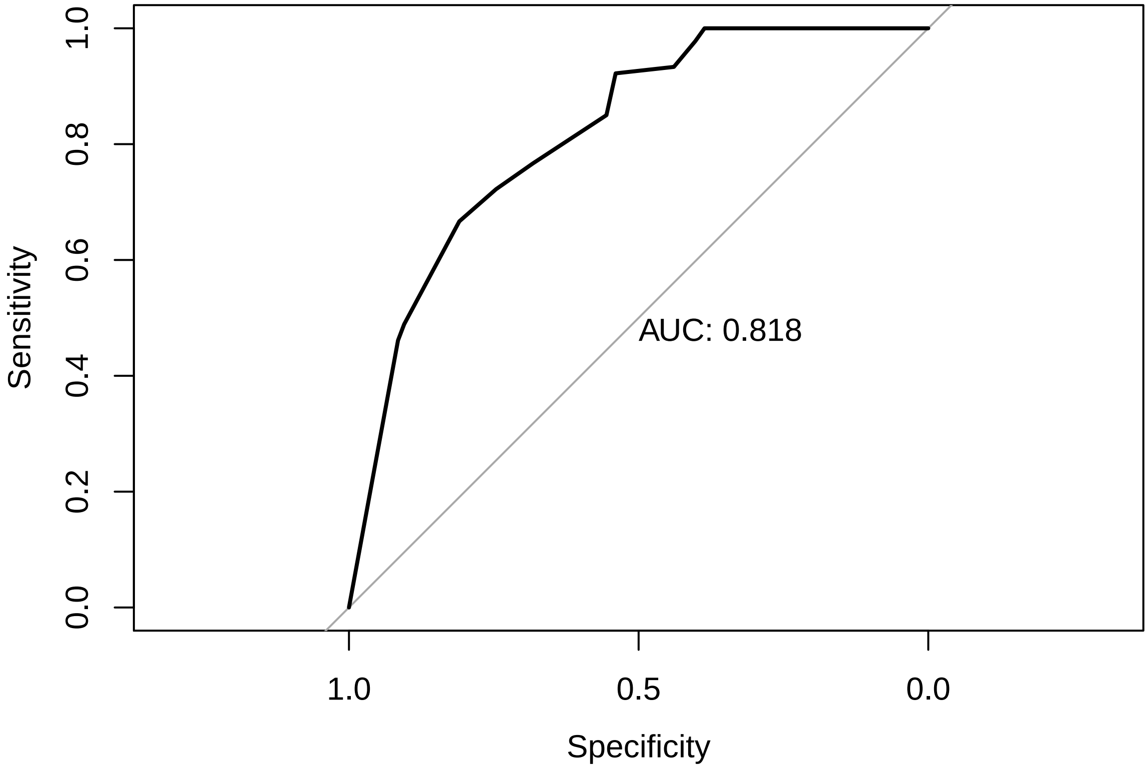

```
##
## Call:
## roc.default(response = nb_pred$truth, predictor = as.numeric(nb_pred$pred),     plot = TRUE, print.au
##
## Data: as.numeric(nb_pred$pred) in 180 controls (nb_pred$truth Allowed) < 189 cases (nb_pred$truth No
## Area under the curve: 0.7235
```

```
#elastic net
roc(response = elas_pred$truth,
               predictor =  as.numeric(elas_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```



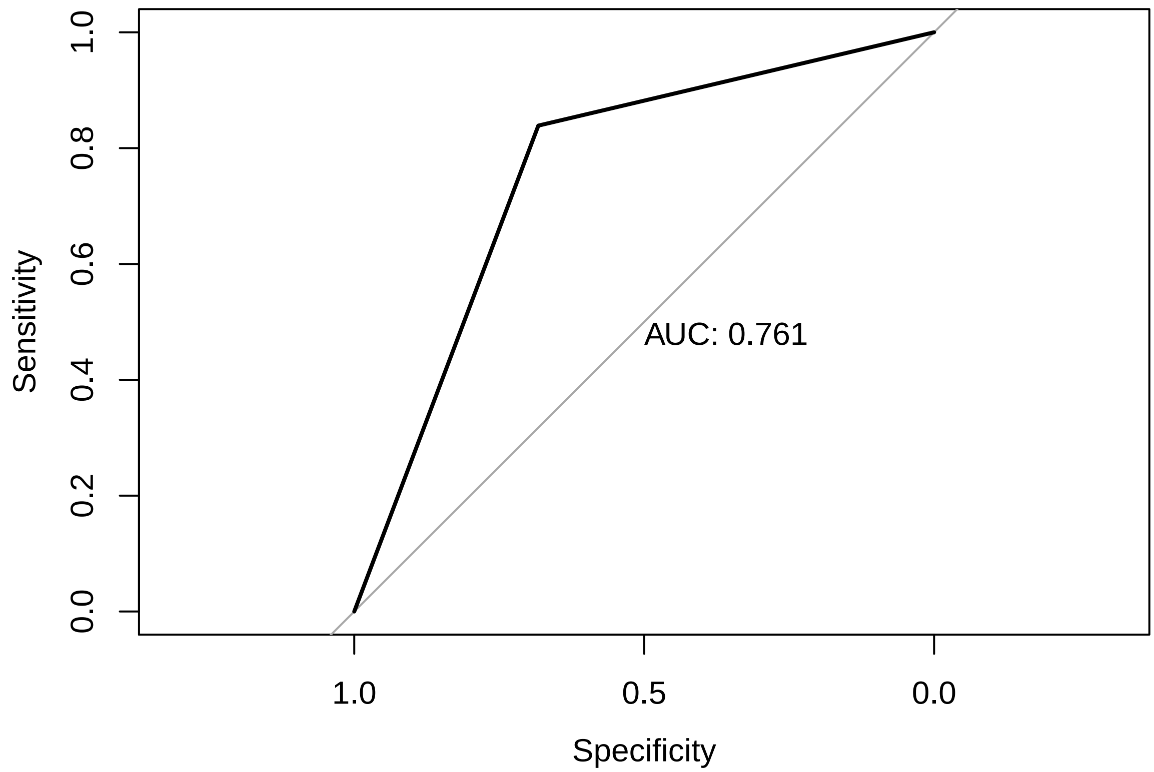

```
##
## Call:
## roc.default(response = elas_pred$truth, predictor = as.numeric(elas_pred$pred),     plot = TRUE, pri
##
## Data: as.numeric(elas_pred$pred) in 189 controls (elas_pred$truth 0) < 180 cases (elas_pred$truth 1)
## Area under the curve: 0.7981
```

```
#decision tree
roc(response = tree_pred$truth,
               predictor =  as.numeric(tree_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```



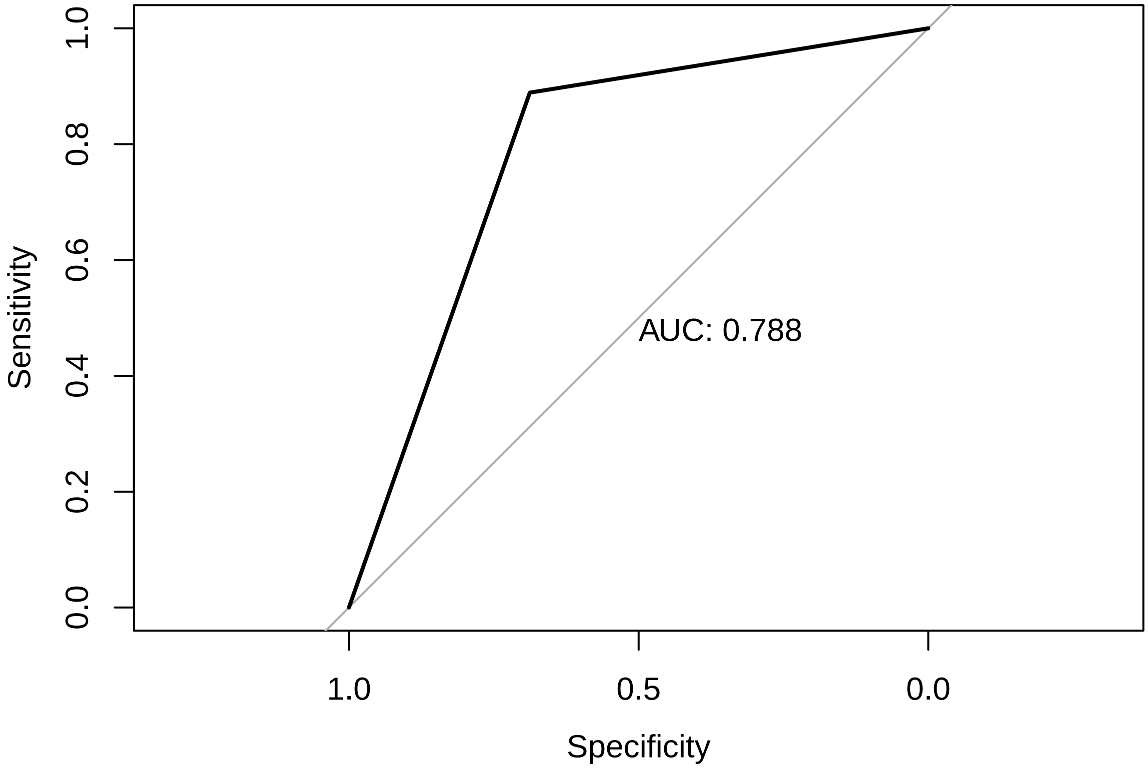

```
##
## Call:
## roc.default(response = tree_pred$truth, predictor = as.numeric(tree_pred$pred),     plot = TRUE, pri
##
## Data: as.numeric(tree_pred$pred) in 189 controls (tree_pred$truth 0) < 180 cases (tree_pred$truth 1)
## Area under the curve: 0.8182
```

```
#bagging
roc(response = bag_pred$truth,
               predictor =  as.numeric(bag_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```



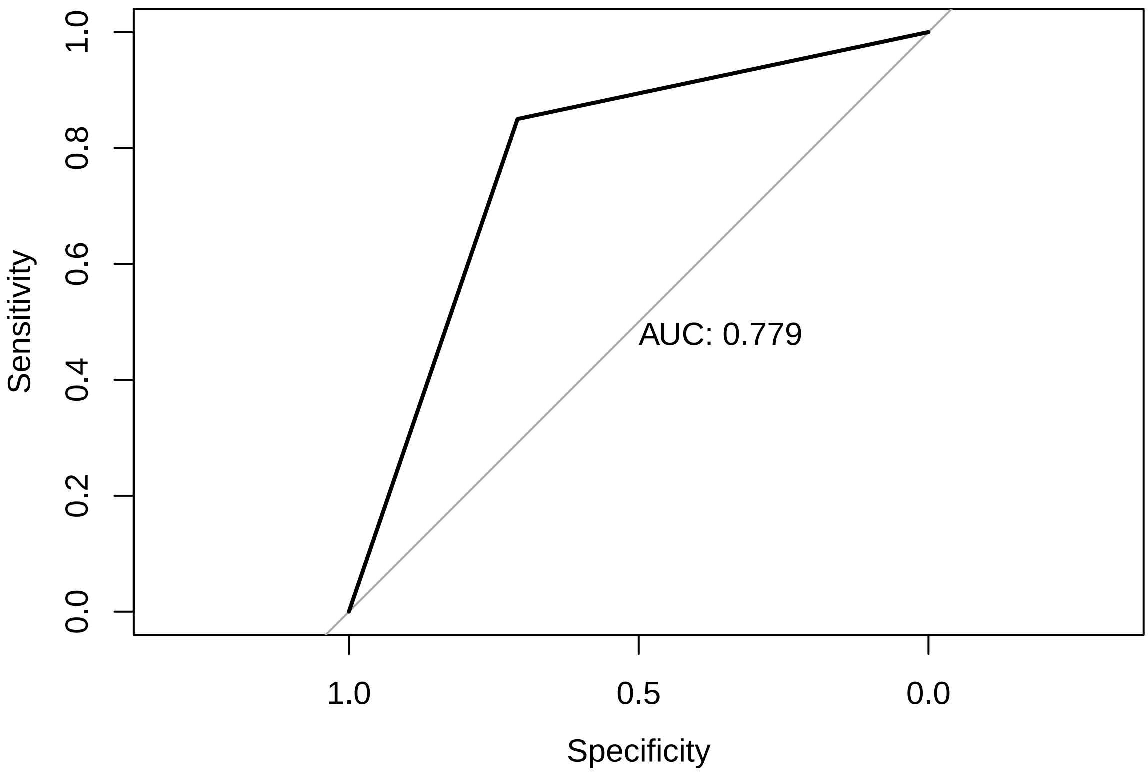

```
##
## Call:
## roc.default(response = bag_pred$truth, predictor = as.numeric(bag_pred$pred),     plot = TRUE, print
##
## Data: as.numeric(bag_pred$pred) in 189 controls (bag_pred$truth 0) < 180 cases (bag_pred$truth 1).
## Area under the curve: 0.7607
```

```
#random forest
roc(response = rf_pred$truth,
               predictor =  as.numeric(rf_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```




```
##
## Call:
## roc.default(response = rf_pred$truth, predictor = as.numeric(rf_pred$pred),     plot = TRUE, print.au
##
## Data: as.numeric(rf_pred$pred) in 189 controls (rf_pred$truth 0) < 180 cases (rf_pred$truth 1).
## Area under the curve: 0.7884
```

```
#boosting
roc(response = boost_pred$truth,
               predictor =  as.numeric(boost_pred$pred),
               plot = TRUE,
               print.auc=TRUE)
```




```
##
## Call:
## roc.default(response = boost_pred$truth, predictor = as.numeric(boost_pred$pred),     plot = TRUE, p
##
## Data: as.numeric(boost_pred$pred) in 189 controls (boost_pred$truth 0) < 180 cases (boost_pred$truth
## Area under the curve: 0.7795
```

### Reasoning

- The best model is the logistic regression model.
- When we compare cross-validated error rates, two models have the lowest number, the logistic regression model (20.596206), and the elastic net model (20.596206). It should be noted that if not specified for binomial response, the error rate of the logistic model will increase by around 2%. Also, if specified for binomial response, the error rate of the elastic net model will be over 50%. We need to compare AUROC to decided which model is better.
- By defition, ROC is plotted with sensitivity (true positive rate) against false positive rate (1 - specificity). Therefore, the higher AUC is, the more accurate a model is in predicting outcomes. Moreover, 1 - AUC is the probability of Type 1 and Type 2 errors. Here, the three highest AUCs belong to logistic regression (0.795), elastic net (0.798), and decision tree (0.818).
- Decision tree is not an option here. When we train the models through cross validation and compare their error rates, it appears decision tree and naive Bayesian have the highest error rate. That means, decision tree is probably doing a bad job in predicting negative outcomes, which produces a high error rate for it.
- The difference between their AUROC is very small (0.003). In this sense, the elastic net model is slightly more accurate than the logistic model in training performance. However, good training performance

sometimes means overfitting, and leads to worse testing performance. To prove which model is better, we need to fit both to the testing set.

## Testing the model

```
X_test <- model.matrix(colrac ~ ., data = gss_test)[, -1]

elas_test <- tibble(truth = gss_test$colrac,
                    pred = round(predict(elas_gss,
                                         s = elas_meters$lambda_1se,
                                         newx = X_test)))

elas_test_err <- elas_test %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

elas_test_err$err_rate
```

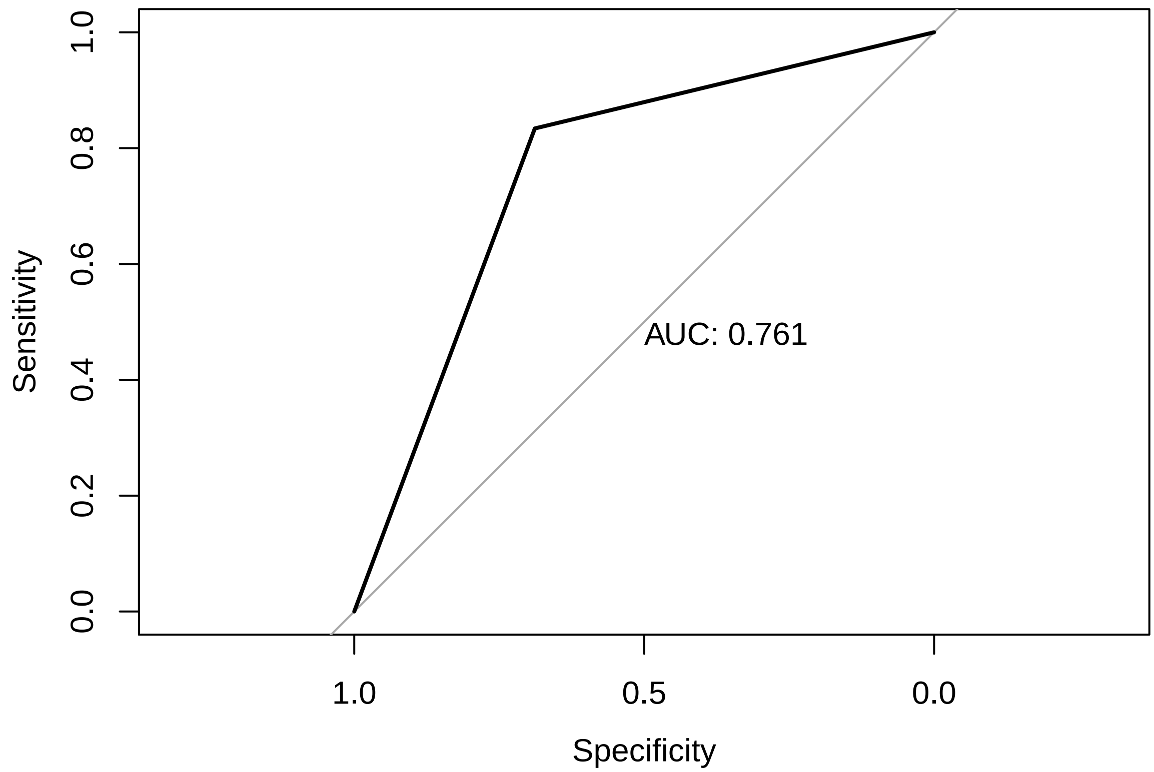
```
## [1] 23.32657
```

```
glm_test <- tibble(truth = gss_test$colrac,
                   pred = round(predict(glm_gss, newdata = gss_test)))

glm_test_err <- glm_test %>%
  count(truth == pred) %>%
  spread("truth == pred", n) %>%
  mutate(err_rate = 100* `FALSE` / (`FALSE` + `TRUE`))

glm_test_err$err_rate
```

```
## [1] 22.10953
```

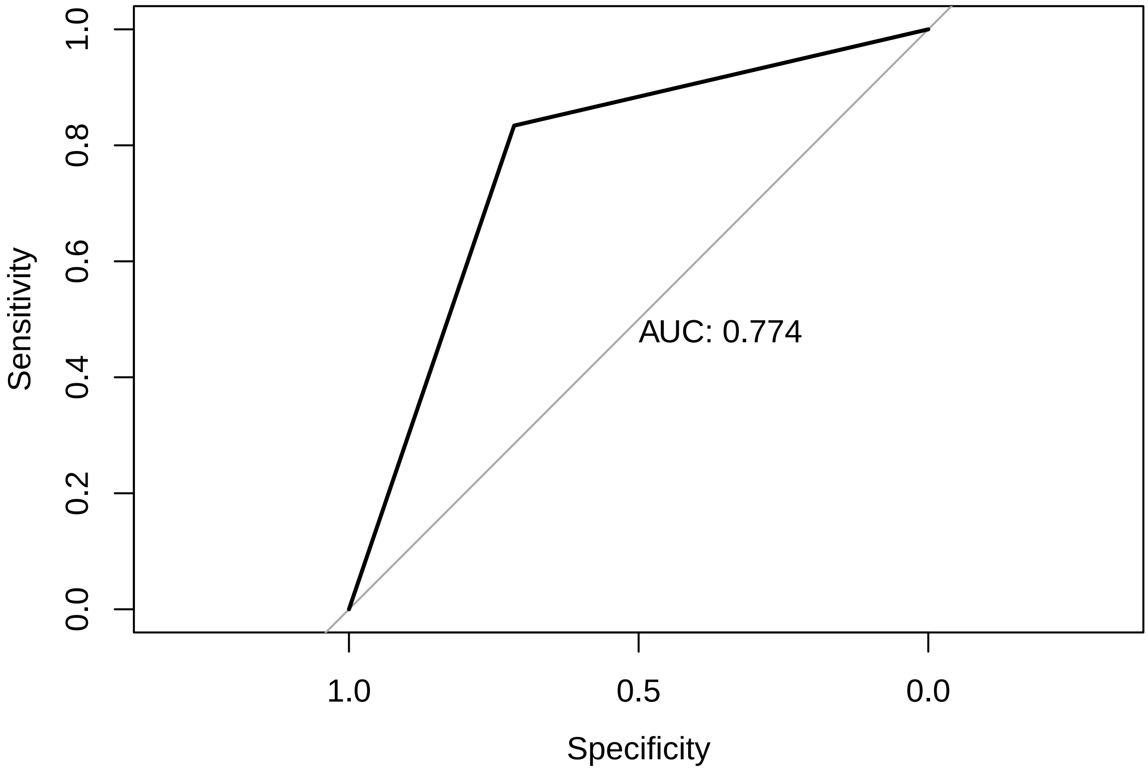
Here, the logistic regression model has a lower error rate than the elastic net model.

```
roc(response = elas_test$truth,
               predictor =  as.numeric(elas_test$pred),
               plot = TRUE,
               print.auc=TRUE)
```




```
##
## Call:
## roc.default(response = elas_test$truth, predictor = as.numeric(elas_test$pred),     plot = TRUE, pri
##
## Data: as.numeric(elas_test$pred) in 228 controls (elas_test$truth 0) < 265 cases (elas_test$truth 1)
## Area under the curve: 0.7613
```

```
roc(response = glm_test$truth,
               predictor =  as.numeric(glm_test$pred),
               plot = TRUE,
               print.auc=TRUE)
```




```
##
## Call:
## roc.default(response = glm_test$truth, predictor = as.numeric(glm_test$pred),     plot = TRUE, print
##
## Data: as.numeric(glm_test$pred) in 228 controls (glm_test$truth 0) < 265 cases (glm_test$truth 1).
## Area under the curve: 0.7744
```

Again, the logistic regression model outperforms the elastic net model in AUROC, by 0.013. Although the difference is still small, it does appear that the logistic regression model is slightly better, and therefore the best among the 7 models.

**Testing performance**

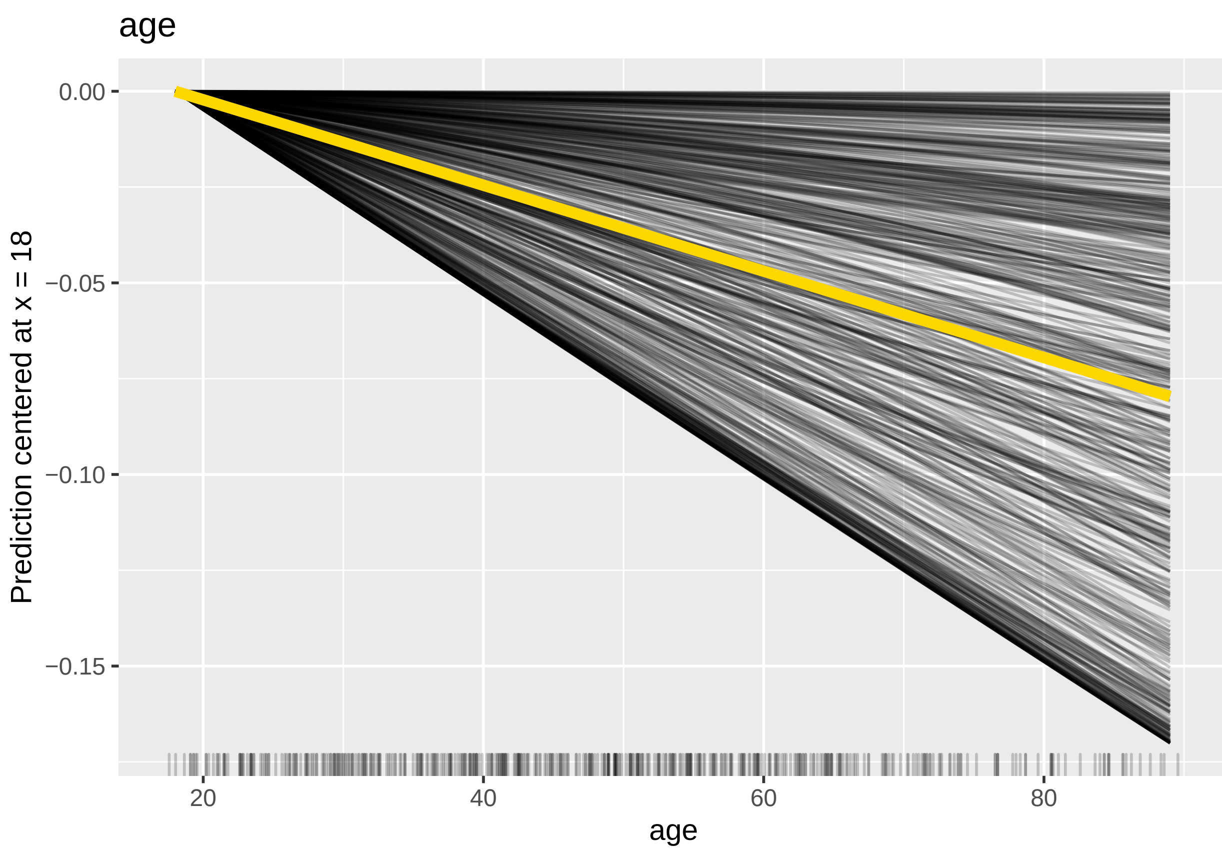
- The logistic regression model performs well on the test set data.
- Its error rate is 22.1095335, which is still lower than 4 out of the 6 remaining models' training error rate. That is, the model testing performance is even better than those model's training performance.
- Moreover, the AUC is 0.774. Compared with its training AUC 0.795, it dropped by 0.021, less than 5%. Considering that the performance is run on a test set, this drop is reasonably small.

## PDP/ICE

**Age**

```r
#separate features from the outcome
features <- gss_test %>%
  dplyr::select(-colrac)

response <- gss_test$colrac

#enter the model and values
predictor.glm <- Predictor$new(
  model = glm_gss,
  data = features,
  y = response,
  predict.fun = predict,
  class = "classification"
)
```

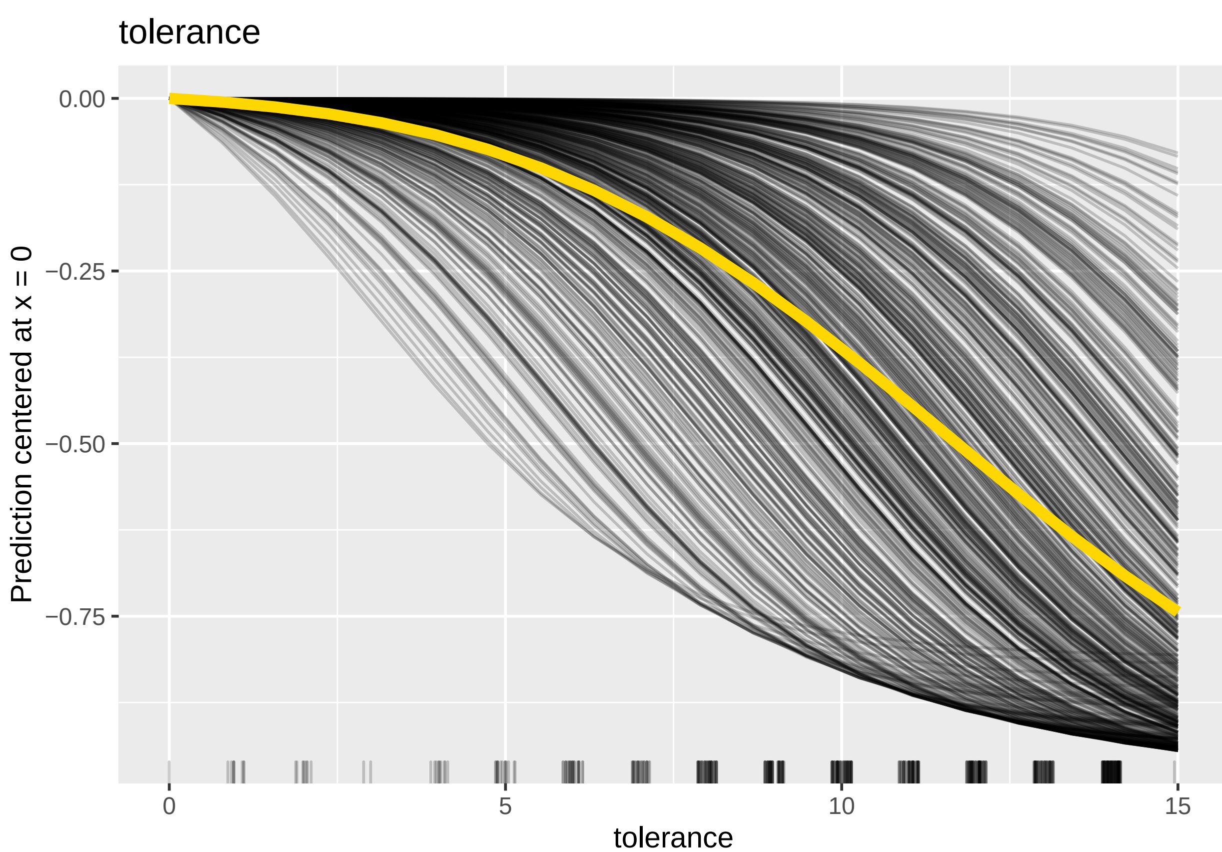
```r
#set up for age
glm.age <- Partial$new(predictor.glm, "age", ice = TRUE)

#set ICE centered on min age
glm.age$center(min(features$age))

#graph ICE plot for age
plot(glm.age) +
  ggtitle("age")
```



The ICE plot centered on min `age`(18) for age shows that in the logistic regression model, `age` is generally negatively associated with the probability of one agreeing to let a racist professor teach in college. The yellow line in the plot (PDP) is the average effect of age on the prediction. Its slope is negative, but very close to 0, which means the impact of `age` on the prediction of `colrac` is very limited.

**Tolerance**

```
#set up for tolerance
glm.tol <- Partial$new(predictor.glm, "tolerance", ice = TRUE)

#set ICE to be centered on min tolerance
glm.tol$center(min(features$tolerance))

#graph ICE plot
plot(glm.tol) +
  ggtitle("tolerance")
```




- The variable `tolerance` is an index based on one's answers to GSS's battery of questions about political tolerance. It is negatively associate with one's tolerance level, i.e., the higher `tolerance` score one has, the less tolerant towards political outgroups the person is.
- In this light, it's no surprise that `tolerance` is negatively associated with the probability of one allowing a racist professor to teach in college. (In fact, `colrac` itself is used to calculate `tolerance`) It's natural that the higher one scores in `tolerance`, the less tolerant he/she is towards outgroups, and the less likely he/she will agree to let a racist professor to teach in college.

- However, the impact of `tolerance` on `colrac` in the model is much stronger than `age`, as we can see by comparing the y-axises. The slope of the yellow line (PDP) in the `tolerance` ICE plot increases as `tolerance` increases. (It's a matter of course, since `colrac` is used to calculate `tolerance`, but not `age`).
- On the other hand, we can see that the majority of `tolerance` scores are between 5 and 14, as shown on the plot. That implies the general population itself is not very "tolerant".