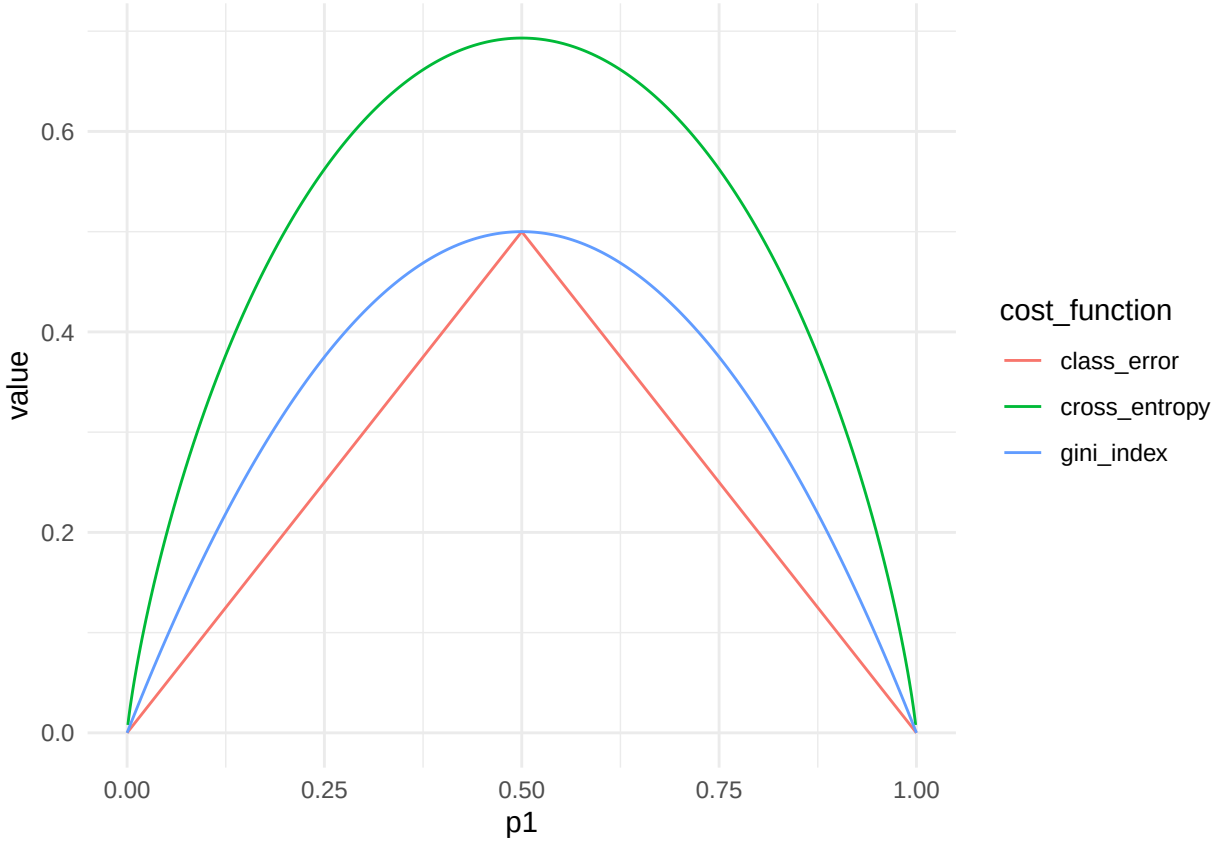
Homework 5: Tree-based Inference

Wanitchaya Poonpatanapricha

1) Conceptual: Cost functions for classification trees

```
p <- seq(0, 1, 0.001)
gini_index <- 2 * p * (1 - p)
class_error <- 1 - pmax(p, 1 - p)
cross_entropy <- -(p * log(p) + (1 - p) * log(1 - p))
cbind(p, gini_index, class_error, cross_entropy) %>% as.data.frame() %>% gather(cost_function,
    value, -p) %>% ggplot(aes(p, value, color = cost_function)) + geom_line() + labs(x = "p1")
```



The graph shows the value for each cost funtion across p_1 from 0 to 1. Since this is a binary classification, p_2 is simply $1 - p_1$. From the graph, we can see that cross entropy is much more sensitive to node impurity than gini index and class error. That is, at any point where p_1 is not 0 or 1 and thus the split will yield impure nodes, cross entropy yields the highest cost value, followed by gini index. On the other hand, class error is very insensitive to node impurity. We can see that the cost value by class error only linearly increases as impurity increases and always remains less than cost values by cross entropy and gini index. *Hence, for growing a decision tree, we should prefer cross entropy > gini index > class error.*

In terms of tree pruning, the appropriate cost function depends on the goal or the success metric of the modeling. In general, we try to maximize accuracy of the prediction from the decision tree. *Hence, it is appropriate to prune the tree using error rate*, which is most directly related to accuracy, over gini index and cross entropy.

Application: Predicting attitudes towards racist college professors

```
train <- read_csv("data/gss_train.csv") %>% mutate(colrac = as.factor(colrac)) %>%
    drop_na()
test <- read_csv("data/gss_test.csv") %>% mutate(colrac = as.factor(colrac)) %>%
    drop_na()
```

2) Estimate the models

```
cv_10 = trainControl(method = "cv", number = 10)
train.matrix = as.matrix(train %>% select(-colrac))
mode(train.matrix) = "numeric"
test.matrix = as.matrix(test %>% select(-colrac))
mode(test.matrix) = "numeric"
```

Logistic Regression

```
# logit
(logit_final <- train(x = train.matrix, y = train$colrac, 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: 1329, 1328, 1328, 1329, 1328, 1328, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.7954    0.5897
```

Naive Bayes

```
# nb
nb_grid <- expand.grid(usekernel = c(TRUE, FALSE), fL = 0:5, adjust = seq(0, 5, by = 2))
nb <- train(x = train.matrix, y = train$colrac, method = "nb", trControl = cv_10,
    tuneGrid = nb_grid)
nb
```

```
## Naive Bayes
##
## 1476 samples
##   55 predictor
```

```
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1328, 1329, 1328, 1329, 1328, ...
## Resampling results across tuning parameters:
##
##   usekernel  fL  adjust  Accuracy  Kappa
##   FALSE      0   0       0.7385    0.4772
##   FALSE      0   2       0.7385    0.4772
##   FALSE      0   4       0.7385    0.4772
##   FALSE      1   0       0.7385    0.4772
##   FALSE      1   2       0.7385    0.4772
##   FALSE      1   4       0.7385    0.4772
##   FALSE      2   0       0.7385    0.4772
##   FALSE      2   2       0.7385    0.4772
##   FALSE      2   4       0.7385    0.4772
##   FALSE      3   0       0.7385    0.4772
##   FALSE      3   2       0.7385    0.4772
##   FALSE      3   4       0.7385    0.4772
##   FALSE      4   0       0.7385    0.4772
##   FALSE      4   2       0.7385    0.4772
##   FALSE      4   4       0.7385    0.4772
##   FALSE      5   0       0.7385    0.4772
##   FALSE      5   2       0.7385    0.4772
##   FALSE      5   4       0.7385    0.4772
##    TRUE      0   0          NaN       NaN
##    TRUE      0   2       0.7243    0.4503
##    TRUE      0   4       0.7270    0.4562
##    TRUE      1   0          NaN       NaN
##    TRUE      1   2       0.7243    0.4503
##    TRUE      1   4       0.7270    0.4562
##    TRUE      2   0          NaN       NaN
##    TRUE      2   2       0.7243    0.4503
##    TRUE      2   4       0.7270    0.4562
##    TRUE      3   0          NaN       NaN
##    TRUE      3   2       0.7243    0.4503
##    TRUE      3   4       0.7270    0.4562
##    TRUE      4   0          NaN       NaN
##    TRUE      4   2       0.7243    0.4503
##    TRUE      4   4       0.7270    0.4562
##    TRUE      5   0          NaN       NaN
##    TRUE      5   2       0.7243    0.4503
##    TRUE      5   4       0.7270    0.4562
##
## 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

```
# elastic net
elnet <- train(x = train.matrix, y = train$colrac, method = "glmnet", trControl = cv_10,
    tuneLength = 5, verbose = FALSE)
```

```
elnet
```

```
## glmnet
##
## 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:
##
##   alpha  lambda     Accuracy  Kappa
##   0.100  0.0002448  0.7954    0.5899
##   0.100  0.0011364  0.7961    0.5912
##   0.100  0.0052748  0.7974    0.5939
##   0.100  0.0244835  0.8008    0.6007
##   0.100  0.1136423  0.7954    0.5895
##   0.325  0.0002448  0.7954    0.5899
##   0.325  0.0011364  0.7967    0.5926
##   0.325  0.0052748  0.7981    0.5952
##   0.325  0.0244835  0.7961    0.5909
##   0.325  0.1136423  0.7757    0.5503
##   0.550  0.0002448  0.7954    0.5899
##   0.550  0.0011364  0.7961    0.5912
##   0.550  0.0052748  0.8001    0.5991
##   0.550  0.0244835  0.7981    0.5945
##   0.550  0.1136423  0.7805    0.5588
##   0.775  0.0002448  0.7961    0.5912
##   0.775  0.0011364  0.7954    0.5898
##   0.775  0.0052748  0.7981    0.5951
##   0.775  0.0244835  0.7981    0.5945
##   0.775  0.1136423  0.7818    0.5613
##   1.000  0.0002448  0.7961    0.5912
##   1.000  0.0011364  0.7947    0.5884
##   1.000  0.0052748  0.7974    0.5937
##   1.000  0.0244835  0.7947    0.5875
##   1.000  0.1136423  0.7804    0.5569
##
## 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 = 0.02448.
```

Decision Tree (CART)

```
# CART
cart <- train(train.matrix, train$colrac, method = "rpart", trControl = cv_10, tuneLength = 5)
cart
```

```
## 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, 1328, ...
## Resampling results across tuning parameters:
##
##   cp        Accuracy  Kappa
##   0.007133  0.7825    0.5631
##   0.008559  0.7825    0.5631
##   0.012839  0.7804    0.5573
##   0.023538  0.7635    0.5207
##   0.499287  0.6539    0.2809
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.008559.
```

Bagging

```
# bagging
bag_final <- train(train.matrix, train$colrac, method = "treebag", trControl = cv_10)
bag_final
```

```
## 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, 1328, 1327, 1329, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.7777    0.552
```

Random Forest

```
rf_grid <- expand.grid(splitrule = c("gini"), min.node.size = c(5, 10, 50), mtry = c(5,
    10, 50))
rf <- train(train.matrix, train$colrac, method = "ranger", trControl = cv_10, tuneGrid = rf_grid)
rf
```

```
## Random Forest
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1328, 1329, 1329, 1329, 1328, ...
## Resampling results across tuning parameters:
##
```

```
##   min.node.size  mtry  Accuracy  Kappa
##    5              5    0.7961    0.5890
##    5             10    0.8042    0.6054
##    5             50    0.7955    0.5880
##   10              5    0.8029    0.6027
##   10             10    0.8042    0.6053
##   10             50    0.7968    0.5904
##   50              5    0.7920    0.5806
##   50             10    0.7920    0.5808
##   50             50    0.7914    0.5794
##
## 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 = 10, splitrule = gini
##  and min.node.size = 5.
```

Boosting

```
gbm_grid <- expand.grid(shrinkage = c(0.1, 0.5, 0.8), n.minobsinnode = c(5, 10, 50),
    n.trees = c(50, 100, 200), interaction.depth = c(5, 10))
gbm <- train(train.matrix, train$colrac, method = "gbm", tuneGrid = gbm_grid, trControl = cv_10,
    verbose = FALSE)
(gbm_final <- gbm$finalModel)
```

```
## A gradient boosted model with bernoulli loss function.
## 200 iterations were performed.
## There were 55 predictors of which 55 had non-zero influence.
```

```
gbm
```

```
## Stochastic Gradient Boosting
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy  Kappa
##   0.1         5                  5               50      0.7974    0.5918
##   0.1         5                  5              100      0.7981    0.5938
##   0.1         5                  5              200      0.7920    0.5818
##   0.1         5                 10               50      0.8035    0.6039
##   0.1         5                 10              100      0.7995    0.5961
##   0.1         5                 10              200      0.8008    0.5992
##   0.1         5                 50               50      0.8055    0.6084
##   0.1         5                 50              100      0.8049    0.6071
##   0.1         5                 50              200      0.8062    0.6101
##   0.1        10                  5               50      0.7954    0.5879
##   0.1        10                  5              100      0.7934    0.5840
##   0.1        10                  5              200      0.7954    0.5882
##   0.1        10                 10               50      0.7988    0.5949
```

```
##   0.1        10                 10               100      0.7873    0.5720
##   0.1        10                 10               200      0.7994    0.5970
##   0.1        10                 50                50      0.8022    0.6018
##   0.1        10                 50               100      0.8035    0.6049
##   0.1        10                 50               200      0.7947    0.5874
##   0.5         5                  5                50      0.7662    0.5318
##   0.5         5                  5               100      0.7717    0.5419
##   0.5         5                  5               200      0.7595    0.5173
##   0.5         5                 10                50      0.7628    0.5237
##   0.5         5                 10               100      0.7554    0.5093
##   0.5         5                 10               200      0.7555    0.5100
##   0.5         5                 50                50      0.7785    0.5554
##   0.5         5                 50               100      0.7684    0.5350
##   0.5         5                 50               200      0.7710    0.5404
##   0.5        10                  5                50      0.7649    0.5280
##   0.5        10                  5               100      0.7778    0.5538
##   0.5        10                  5               200      0.7832    0.5640
##   0.5        10                 10                50      0.7574    0.5129
##   0.5        10                 10               100      0.7717    0.5413
##   0.5        10                 10               200      0.7771    0.5520
##   0.5        10                 50                50      0.7690    0.5366
##   0.5        10                 50               100      0.7710    0.5405
##   0.5        10                 50               200      0.7765    0.5509
##   0.8         5                  5                50      0.7406    0.4807
##   0.8         5                  5               100      0.7304    0.4595
##   0.8         5                  5               200      0.7270    0.4537
##   0.8         5                 10                50      0.7432    0.4849
##   0.8         5                 10               100      0.7405    0.4799
##   0.8         5                 10               200      0.7405    0.4794
##   0.8         5                 50                50      0.7534    0.5057
##   0.8         5                 50               100      0.7446    0.4882
##   0.8         5                 50               200      0.7547    0.5084
##   0.8        10                  5                50      0.7344    0.4673
##   0.8        10                  5               100      0.7229    0.4461
##   0.8        10                  5               200      0.6871    0.3723
##   0.8        10                 10                50      0.7568    0.5130
##   0.8        10                 10               100      0.7527    0.5045
##   0.8        10                 10               200      0.7649    0.5288
##   0.8        10                 50                50      0.7561    0.5116
##   0.8        10                 50               100      0.7629    0.5240
##   0.8        10                 50               200      0.7690    0.5367
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 200, interaction.depth =
##  5, shrinkage = 0.1 and n.minobsinnode = 50.
```

3) Evaluate the models

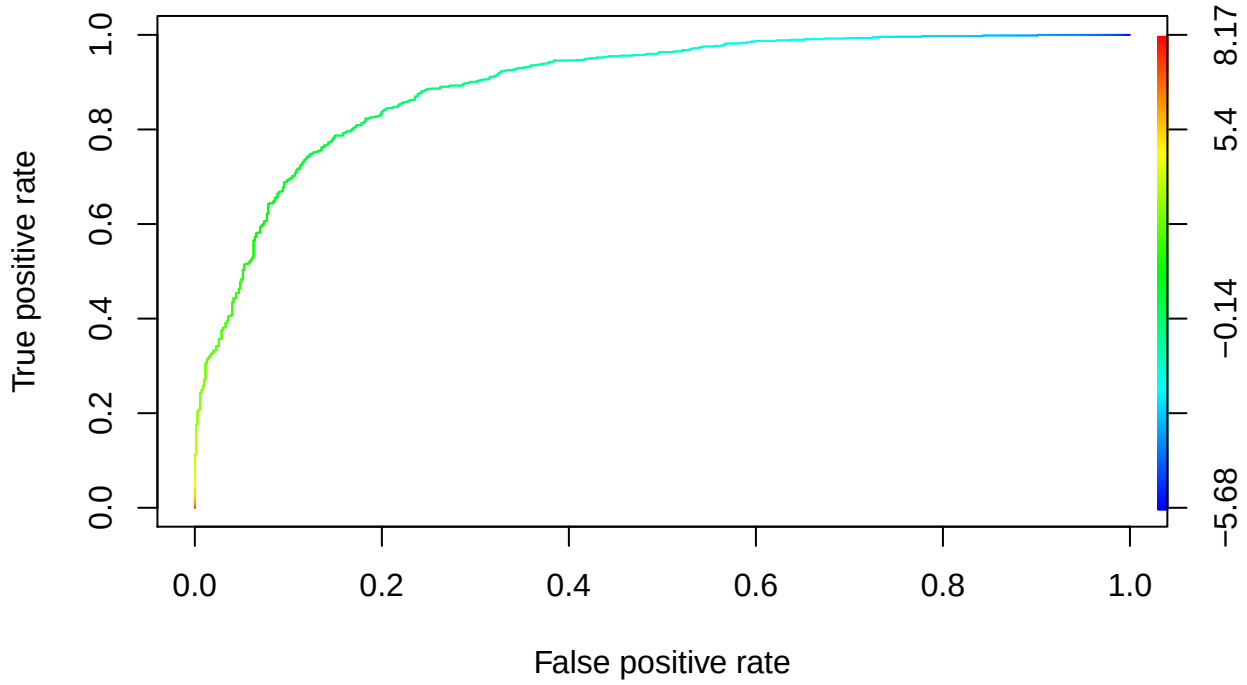
Logistic Regression

The model does pretty well in both error rate and ROC/AUC given that it is just a simple regression.

```
# error
1 - max(logit_final$results$Accuracy)
```

```
## [1] 0.2046
```

```
# roc
pred <- prediction(predict(logit_final$finalModel, newdata = train), train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8965
```

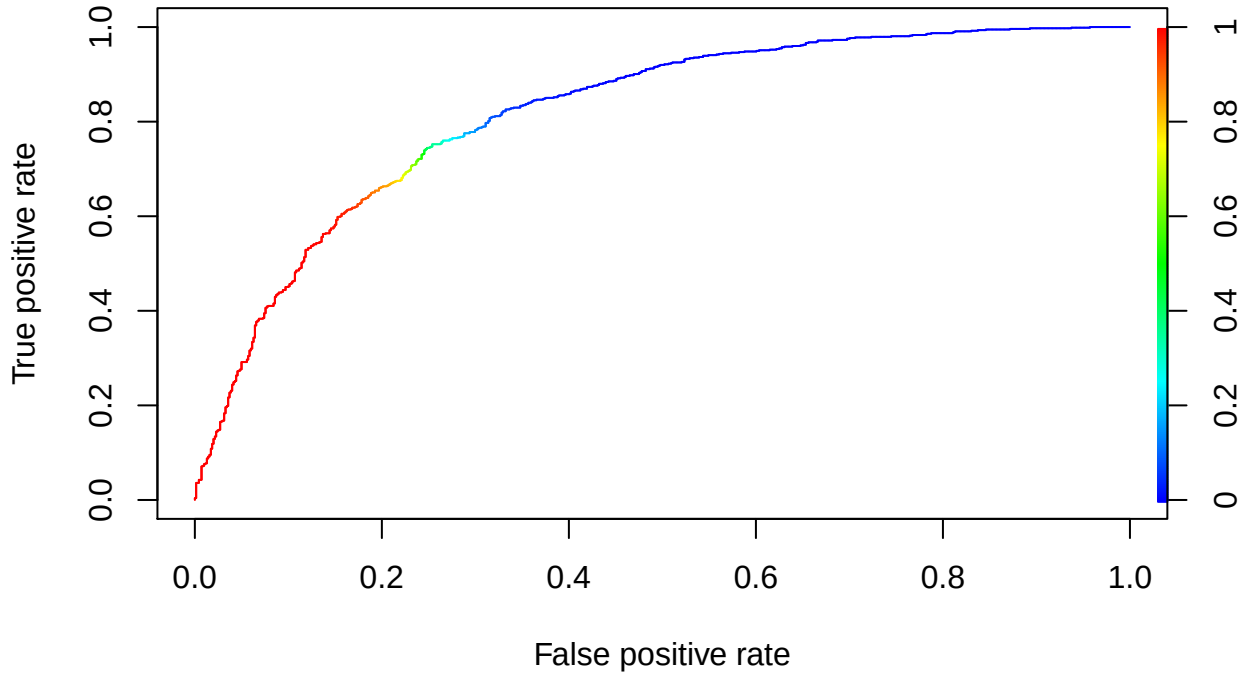
Naive Bayes

The model does significantly worse than logistic regression.

```
# error
1 - max(nb$results$Accuracy, na.rm = T)
```

```
## [1] 0.2615
```

```
# roc
pred <- prediction(predict(nb$finalModel, newdata = train, type = "raw")$posterior[,
    2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8164
```

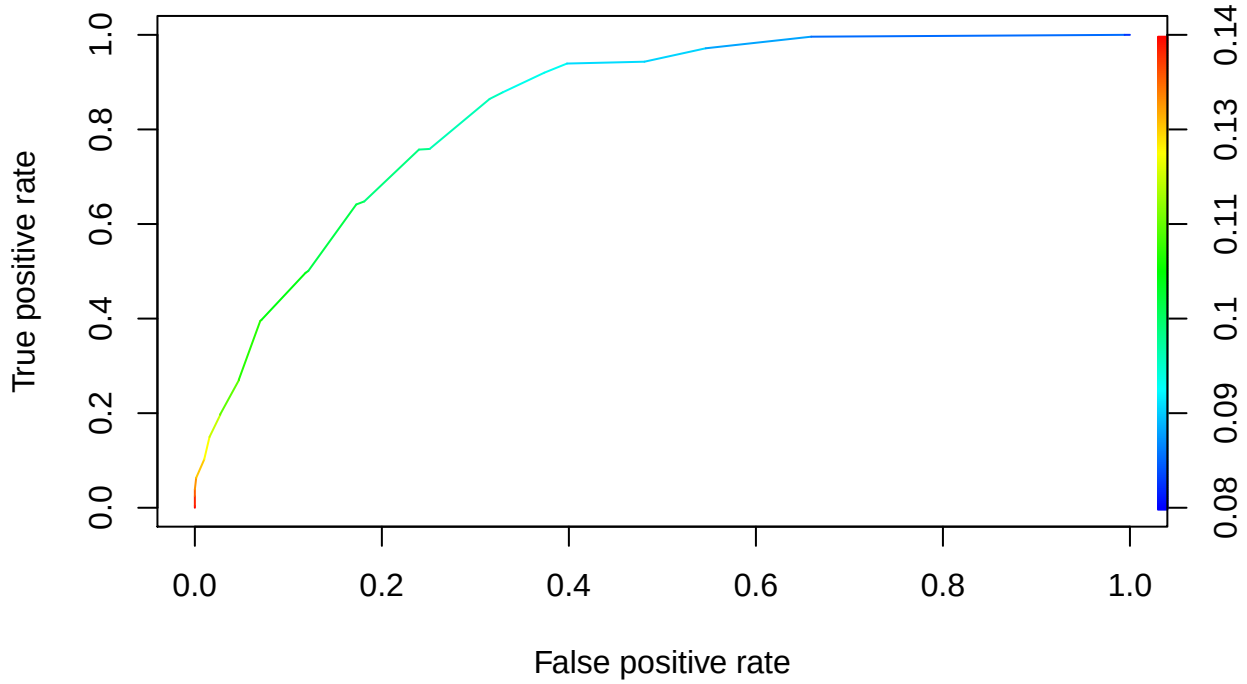
Elastic Net Regression

The model does better than Naive Bayes but still slightly worse than logistive regression.

```
# error
1 - max(elnet$results$Accuracy, na.rm = T)
```

```
## [1] 0.1992
```

```
# roc
pred <- prediction(predict(elnet$finalModel, newx = train.matrix)[, 2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8421
```

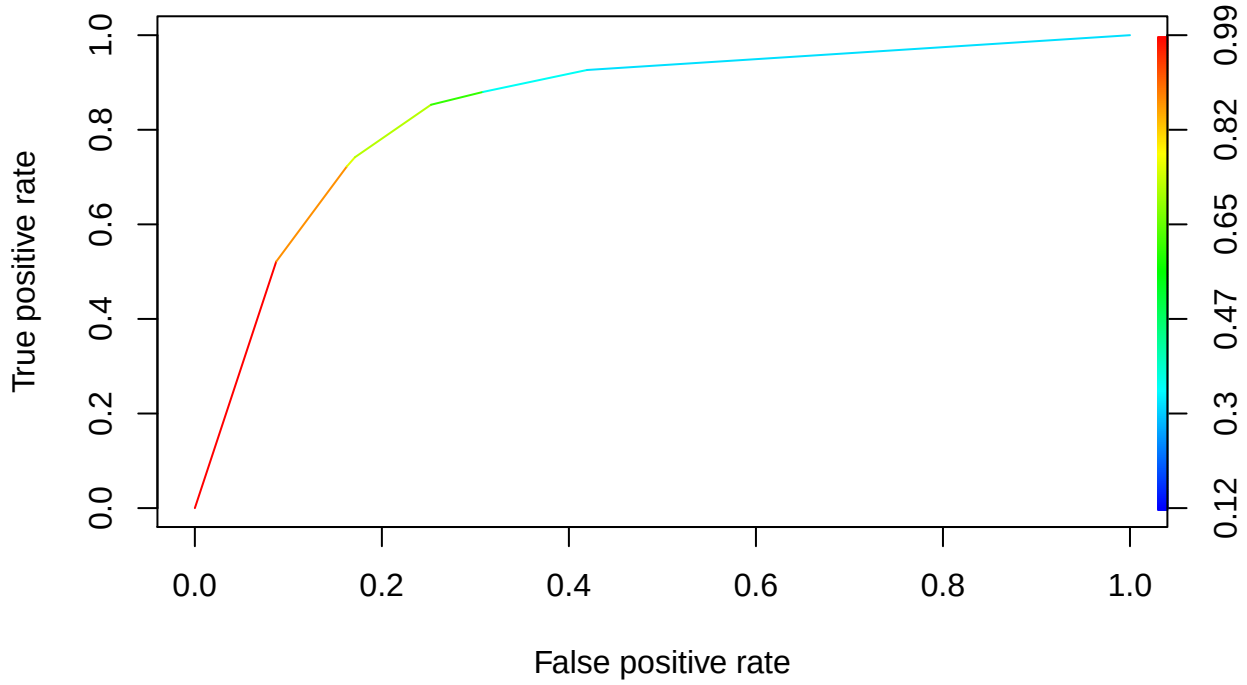
Decision Tree (CART)

The model does slightly worse than Elastic Net but still better than Naive Bayes.

```
# error
1 - max(cart$results$Accuracy, na.rm = T)
```

```
## [1] 0.2175
```

```
# roc
pred <- prediction(predict(cart$finalModel, newdata = train)[, 2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8488
```

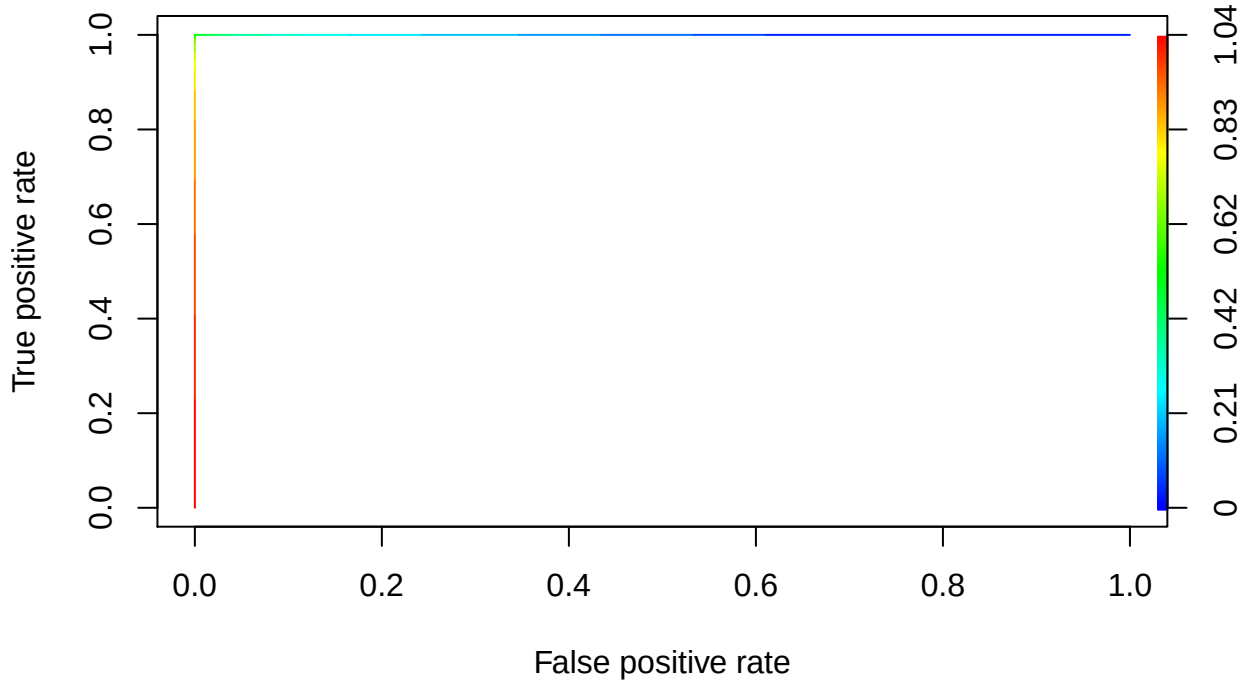
Bagging

The model's ROC/AUC are perfect! However, the error rate is not the best compared to other models.

```
# error
1 - max(bag_final$results$Accuracy, na.rm = T)
```

```
## [1] 0.2223
```

```
# roc
pred <- prediction(predict(bag_final, newdata = train, type = "prob")[, 2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

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

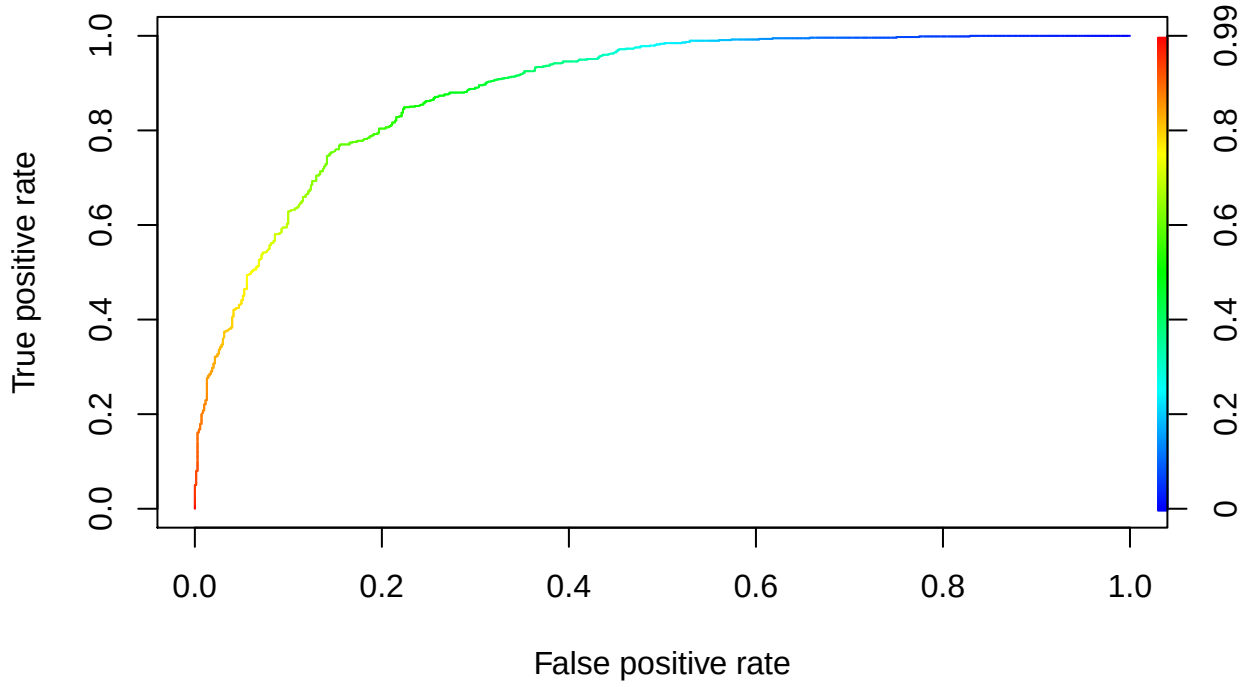
Random Forest

The model does slightly better than logistic regression. However, considering high computing cost of random forest, logistic regression might be a better choice.

```
# error
1 - max(rf$results$Accuracy, na.rm = T)
```

```
## [1] 0.1958
```

```
# roc
rf_final <- randomForest::randomForest(colrac ~ ., data = train, mtry = rf$bestTune$mtry,
    nodesize = rf$bestTune$min.node.size)
pred <- prediction(predict(rf_final, data = train, type = "prob")[, 2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8879
```

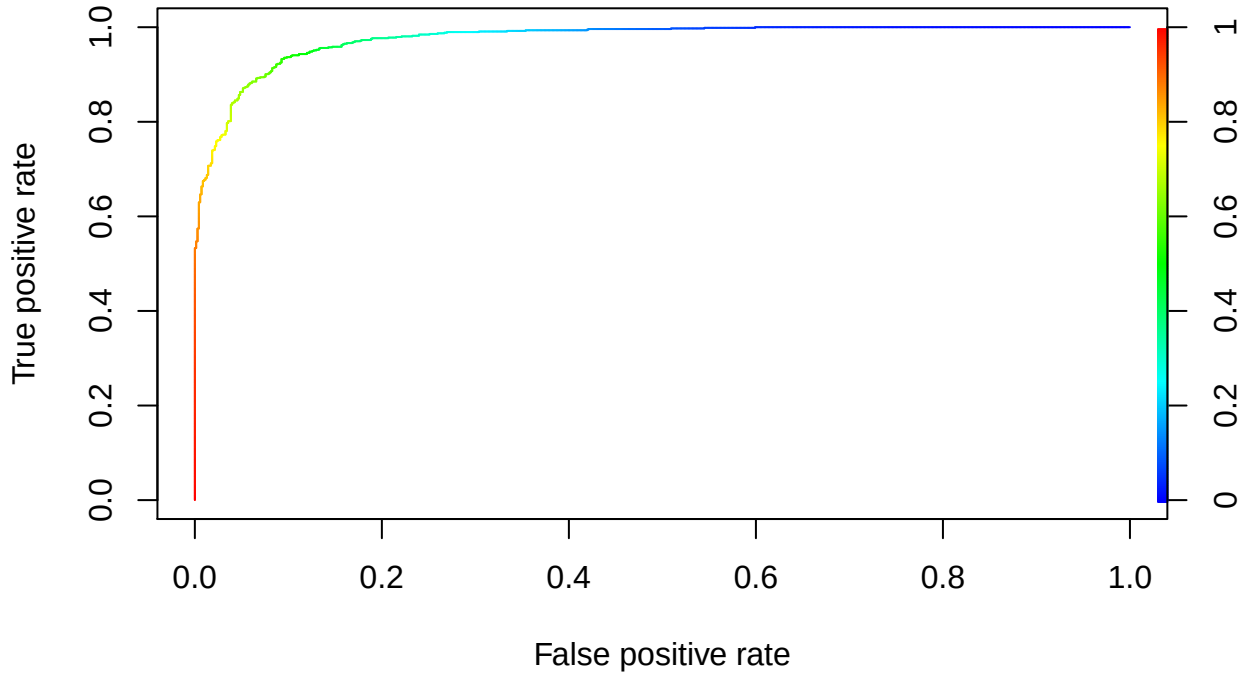
Boosting

Both ROC/AUC and error rate of boosting are really good. This is the best model we see so far!

```
# error
1 - max(gbm$results$Accuracy, na.rm = T)
```

```
## [1] 0.1938
```

```
# roc
pred <- prediction(predict(gbm, newdata = train, type = "prob")[, 2], train$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.9752
```

4) Choose the best model

The two best models according to ROC/AUC are Bagging and Boosting respectively. Bagging in fact has a perfect ROC/AUC. However, Bagging's error rate is almost 3% higher than Boosting's. In addition, Boosting's ROC/AUC are almost perfect. Hence, I believe **Boosting** is the best model considering both ROC/AUC and error rate.

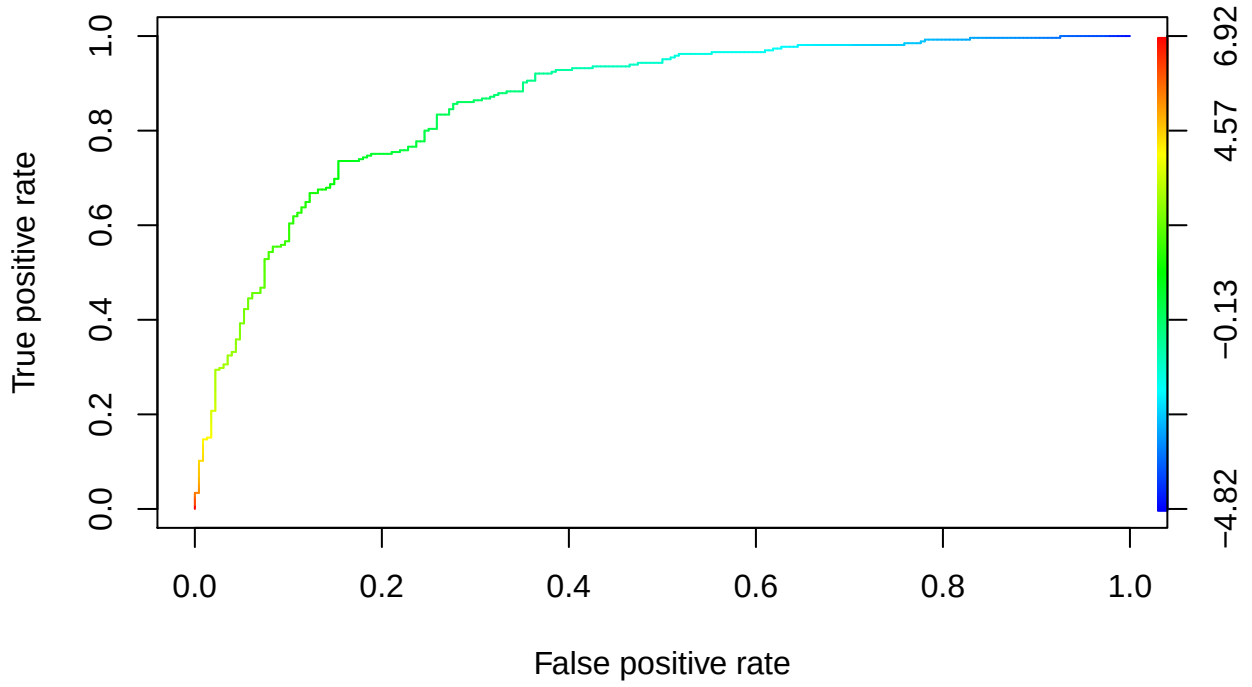
5) Evaluate the *best* model

Logistic Regression

```
# error
1 - max(logit_final$results$Accuracy)
```

```
## [1] 0.2046
```

```
# roc
pred <- prediction(predict(logit_final$finalModel, newdata = test), test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

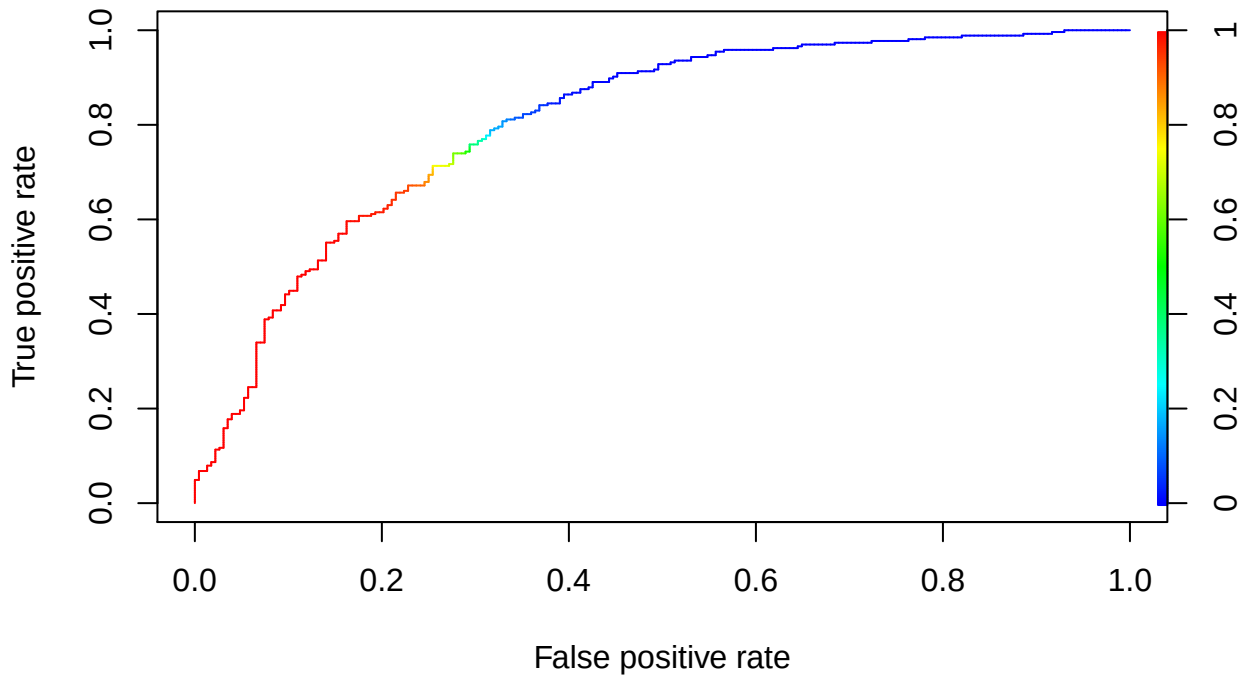
```
## [1] 0.8619
```

Naive Bayes

```
# error
1 - max(nb$results$Accuracy, na.rm = T)
```

```
## [1] 0.2615
```

```
# roc
pred <- prediction(predict(nb$finalModel, newdata = test, type = "raw")$posterior[,
    2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

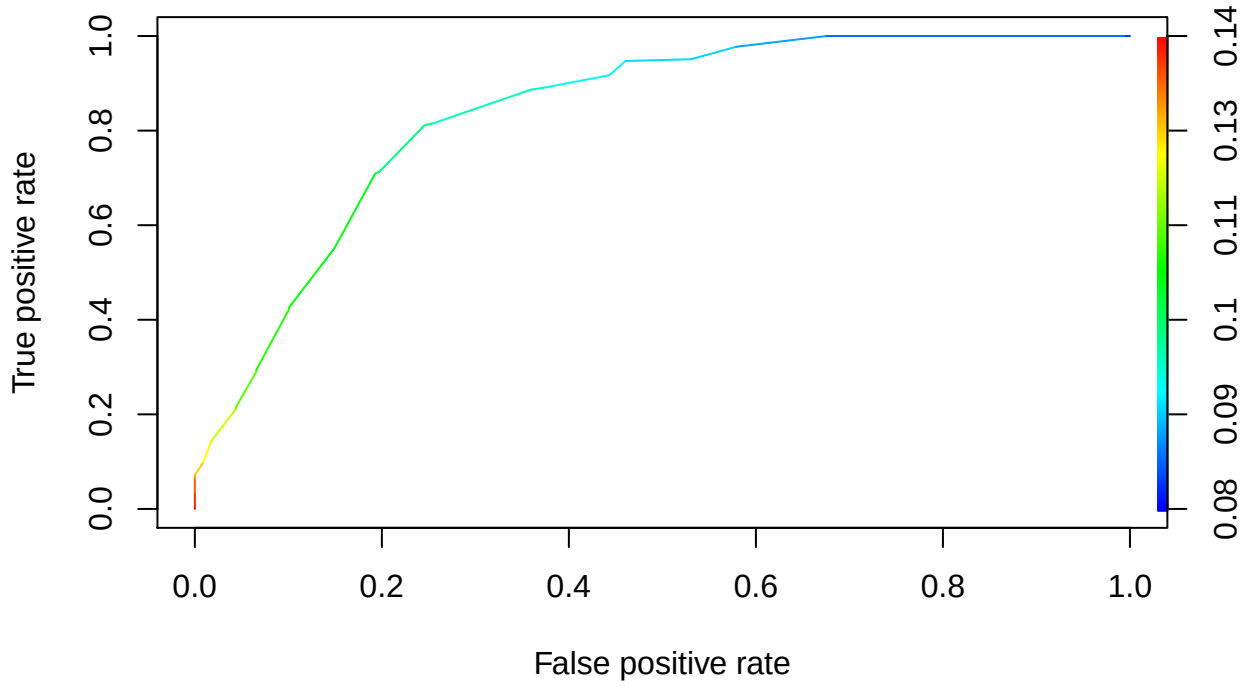
```
## [1] 0.8061
```

Elastic Net Regression

```
# error
1 - max(elnet$results$Accuracy, na.rm = T)
```

```
## [1] 0.1992
```

```
# roc
pred <- prediction(predict(elnet$finalModel, newx = test.matrix)[, 2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

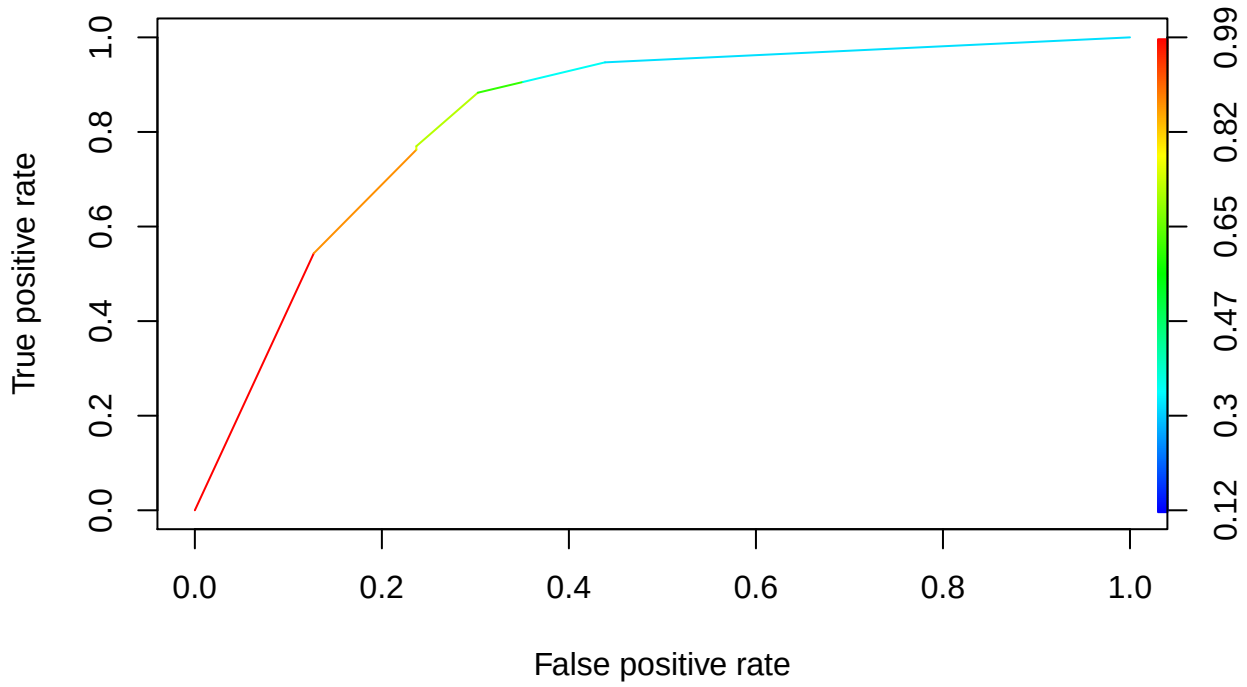
```
## [1] 0.837
```

Decision Tree (CART)

```
# error
1 - max(cart$results$Accuracy, na.rm = T)
```

```
## [1] 0.2175
```

```
# roc
pred <- prediction(predict(cart$finalModel, newdata = test)[, 2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

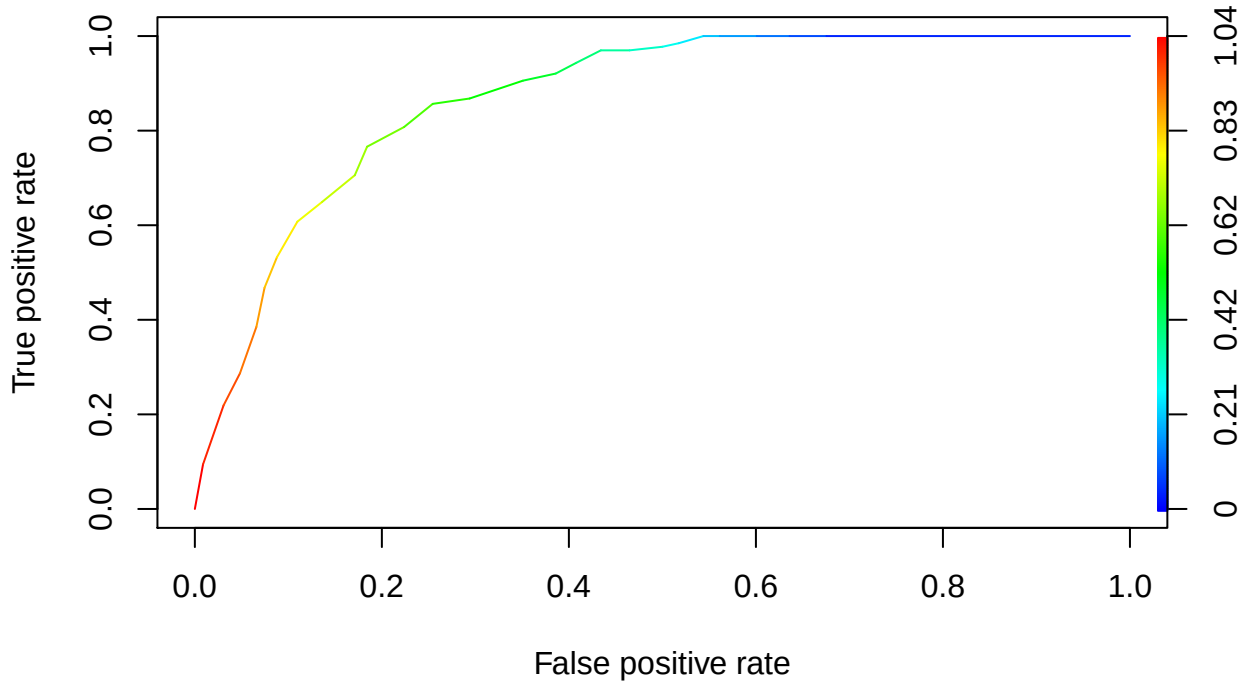
```
## [1] 0.8315
```

Bagging

```
# error
1 - max(bag_final$results$Accuracy, na.rm = T)
```

```
## [1] 0.2223
```

```
# roc
pred <- prediction(predict(bag_final, newdata = test, type = "prob")[, 2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

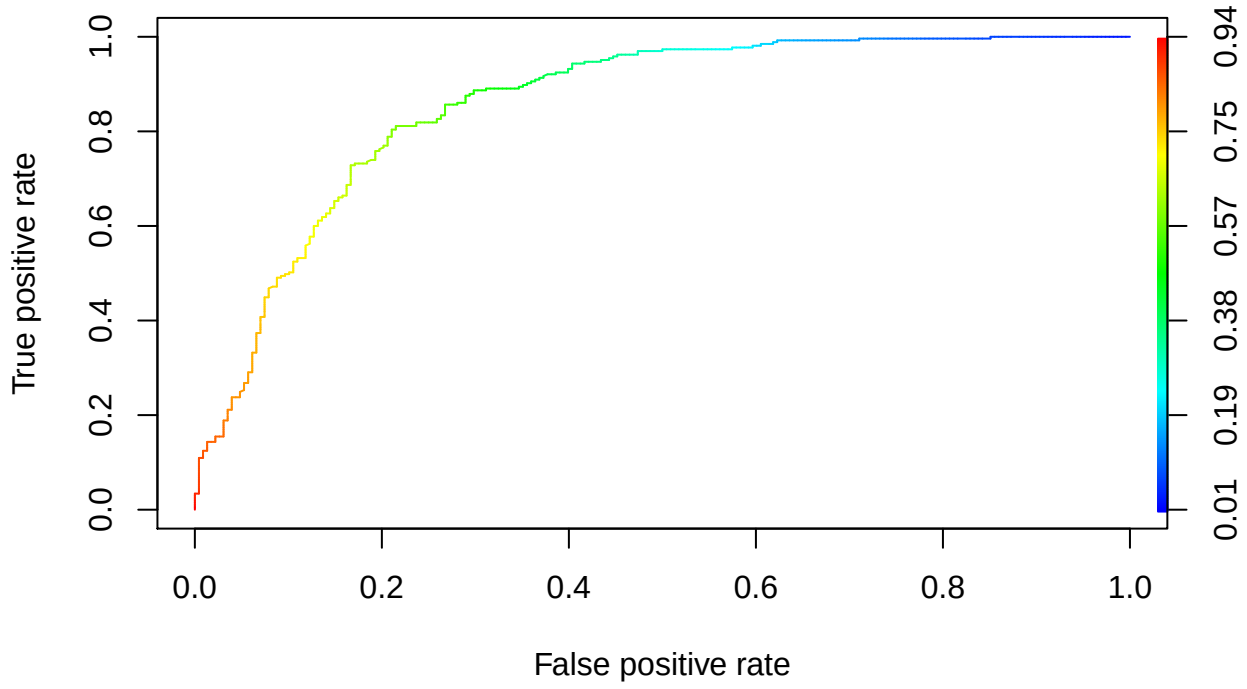
```
## [1] 0.8692
```

Random Forest

```
# error
1 - max(rf$results$Accuracy, na.rm = T)
```

```
## [1] 0.1958
```

```
# roc
rf_final <- randomForest::randomForest(colrac ~ ., data = test, mtry = rf$bestTune$mtry,
    nodesize = rf$bestTune$min.node.size)
pred <- prediction(predict(rf_final, data = test, type = "prob")[, 2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```




```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

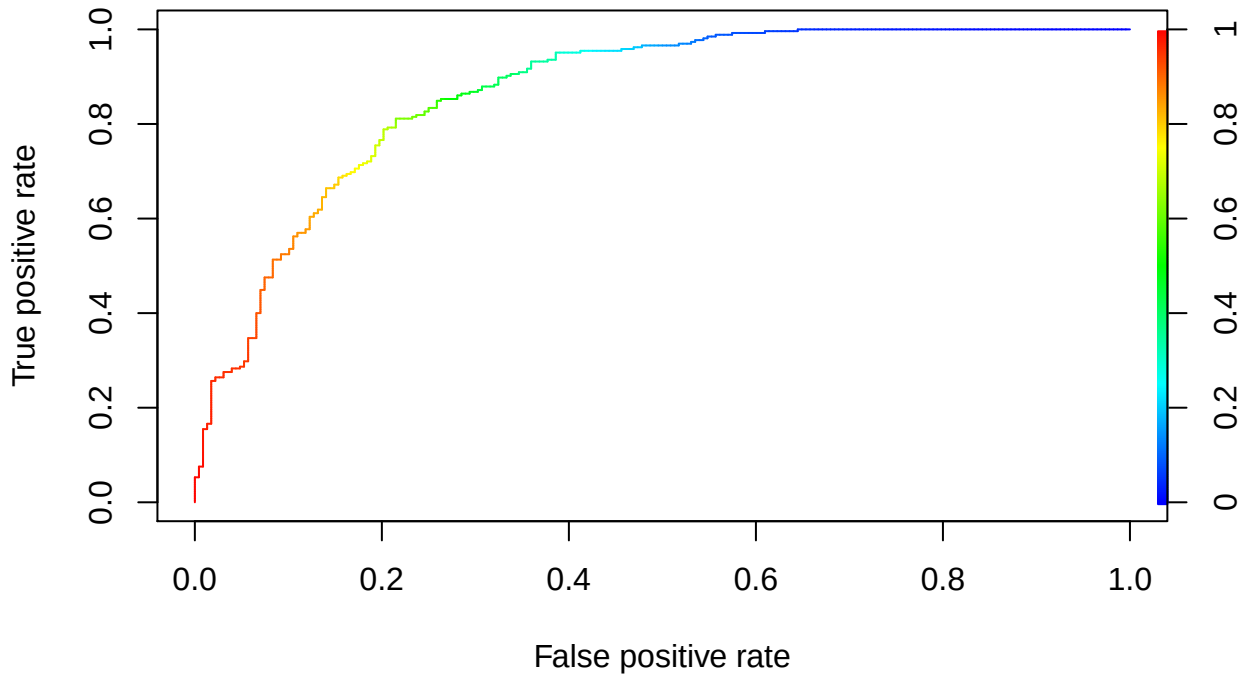
```
## [1] 0.8573
```

Boosting

```
# error
1 - max(gbm$results$Accuracy, na.rm = T)
```

```
## [1] 0.1938
```

```
# roc
pred <- prediction(predict(gbm, newdata = test, type = "prob")[, 2], test$colrac)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)
```



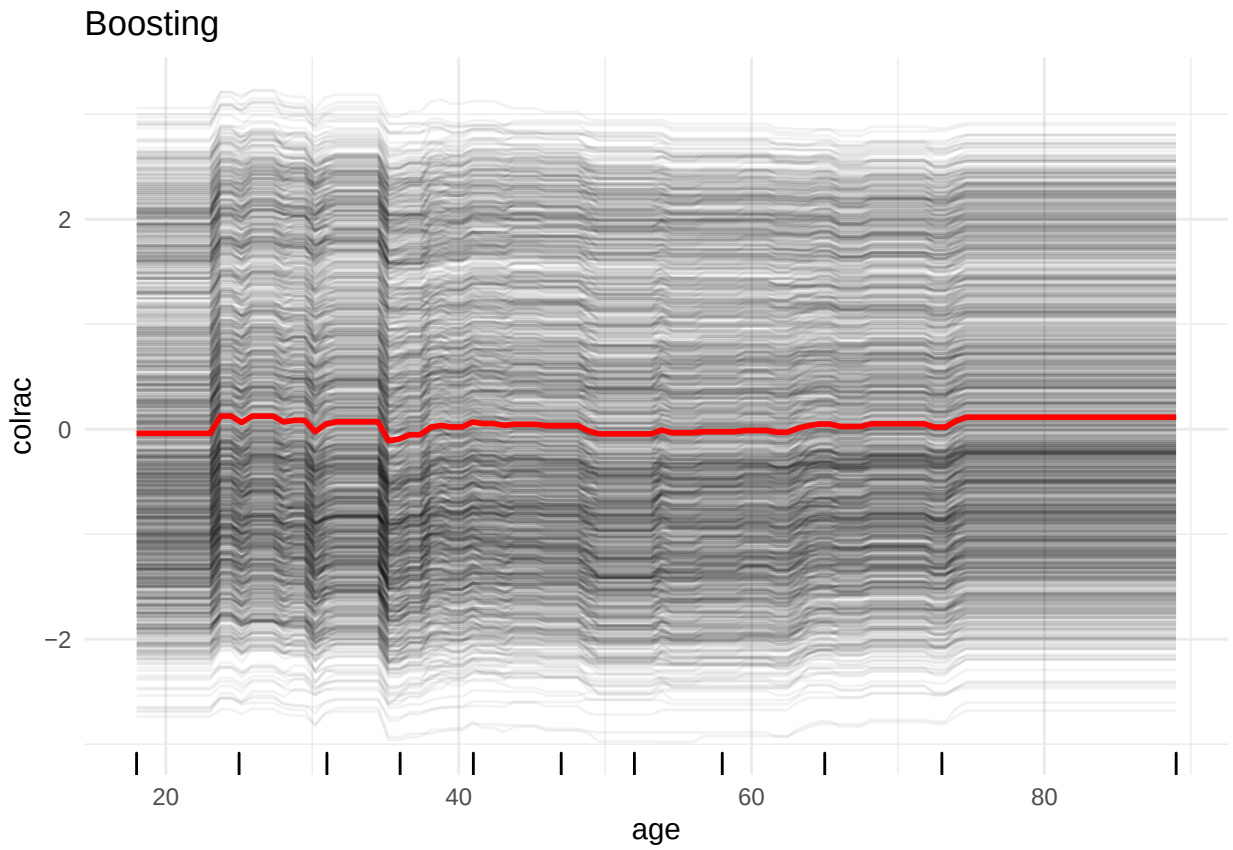

```
# auc
performance(pred, measure = "auc")@y.values[[1]]
```

```
## [1] 0.8661
```

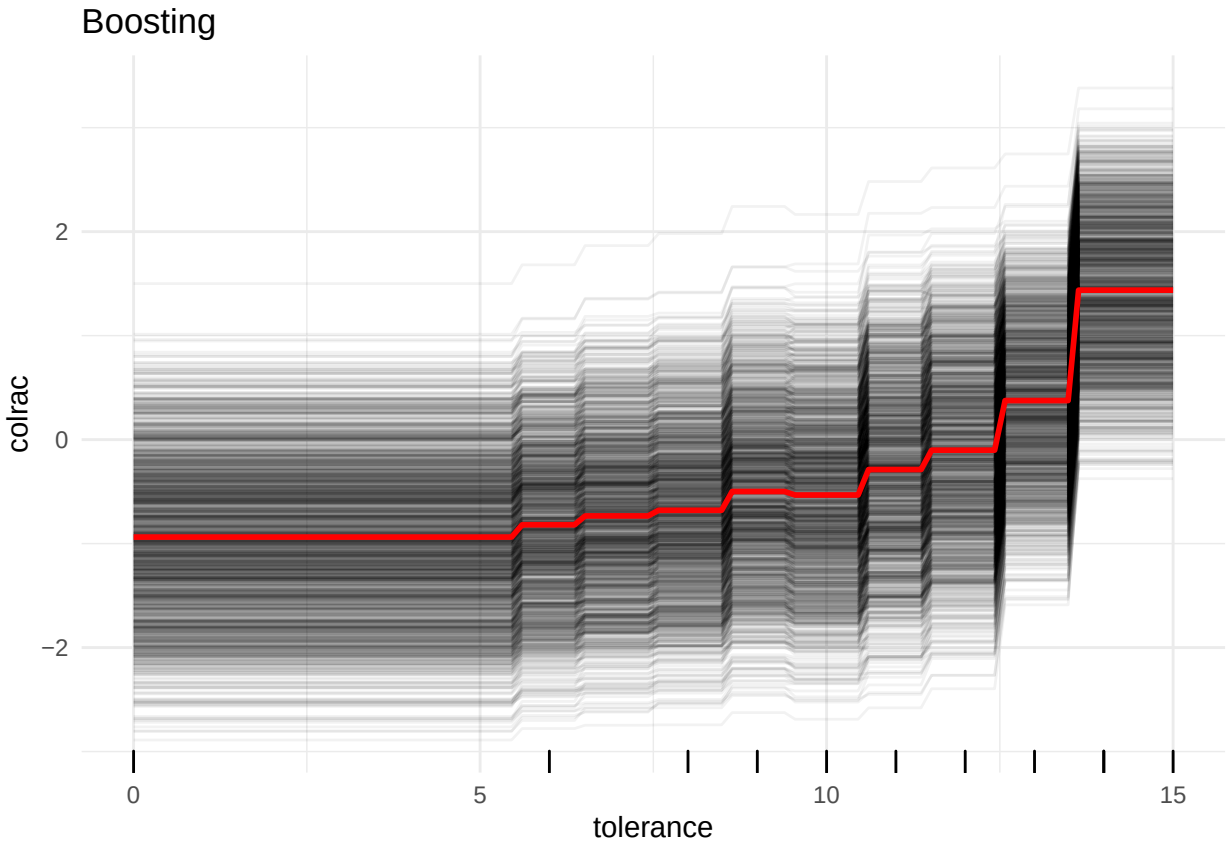
Most of the models have similar test error rate and test ROC/AUC to their train counterparts. This indicates that most of the models are not overfitted to the training data, which is desirable. This is because we use cross-validation in training. Boosting and Bagging, however, have significant drop in test ROC/AUC from their train ROC/AUC. Their test ROC/AUC drop from perfect in train to around 86, which is similar to test ROC/AUC of logistic regression. The test error rates of logistic regression and boosting are also very close. Hence, according to the model performances in test session, the best model in my opinion is **logistic regression** over boosting. This is because logistic regression requires a lot less computing resources and is way easier to interpret while still performs almost as well as boosting in test session.

6) Bonus: PDPs/ICE

```
library(pdp)
gbm %>% partial(pred.var = "age", grid.resolution = 100, ice = TRUE) %>% autoplot(rug = TRUE,
    train = train, alpha = 0.05) + labs(title = "Boosting", y = "colrac")
```




```
gbm %>% partial(pred.var = "tolerance", grid.resolution = 100, ice = TRUE) %>% autoplot(rug = TRUE,
    train = train, alpha = 0.05) + labs(title = "Boosting", y = "colrac")
```

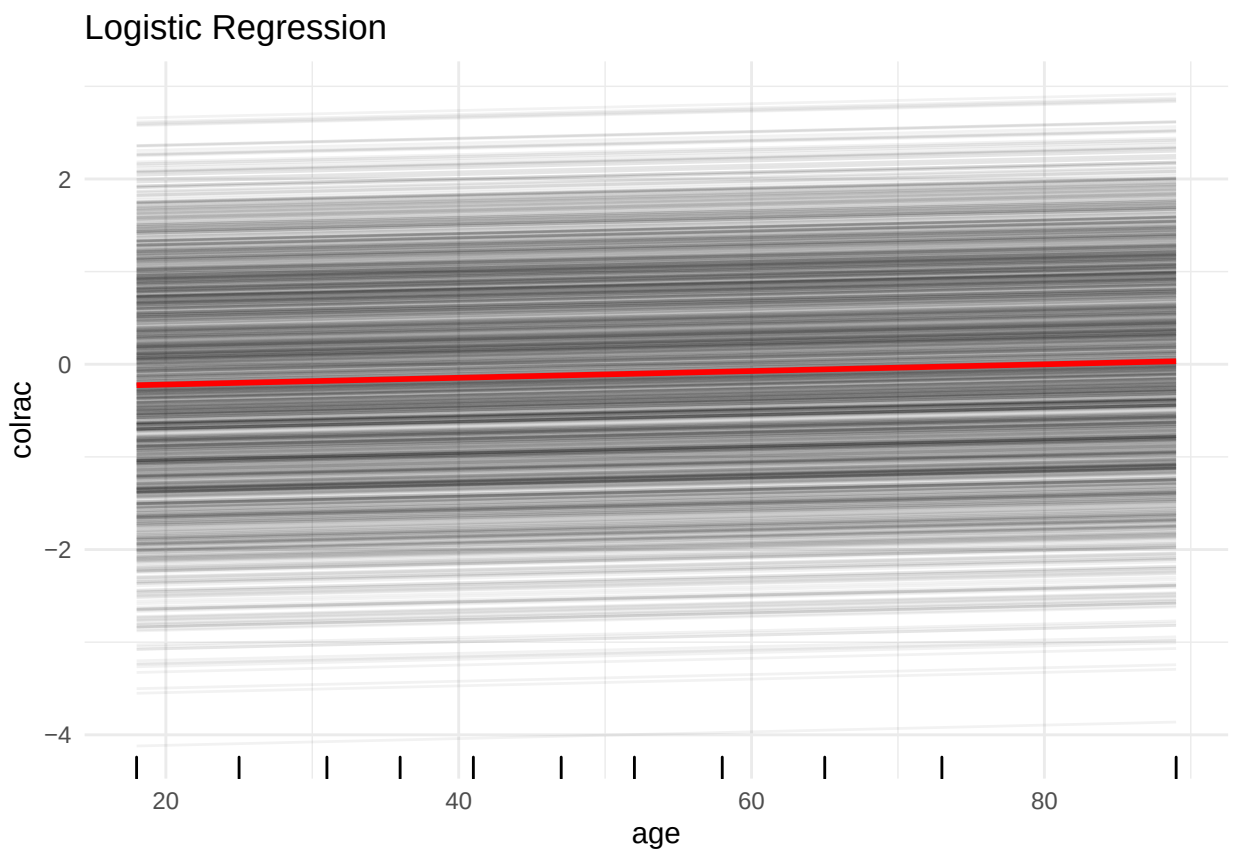


From 4), I choose **Boosting** as the best model (given training session alone). From `age`'s PDP/ICE, we can see that there is not much influence of `age` on `colrac`. Also, there doesn't seem to be any outlier regarding to `age` from the ICE. The general shape of the `age`'s PDP/ICE is not smooth. This makes sense because boosting is essentially a type of decision tree, and decision tree generally devide the predictor's space into sub-regions. However, since each sub-region of `age` has pretty much the same prediction of `colrac` to one another (on top of that, the prediction is around 0), there doesn't seem to be a substantive reason to including `age` as one of the predictors at all.

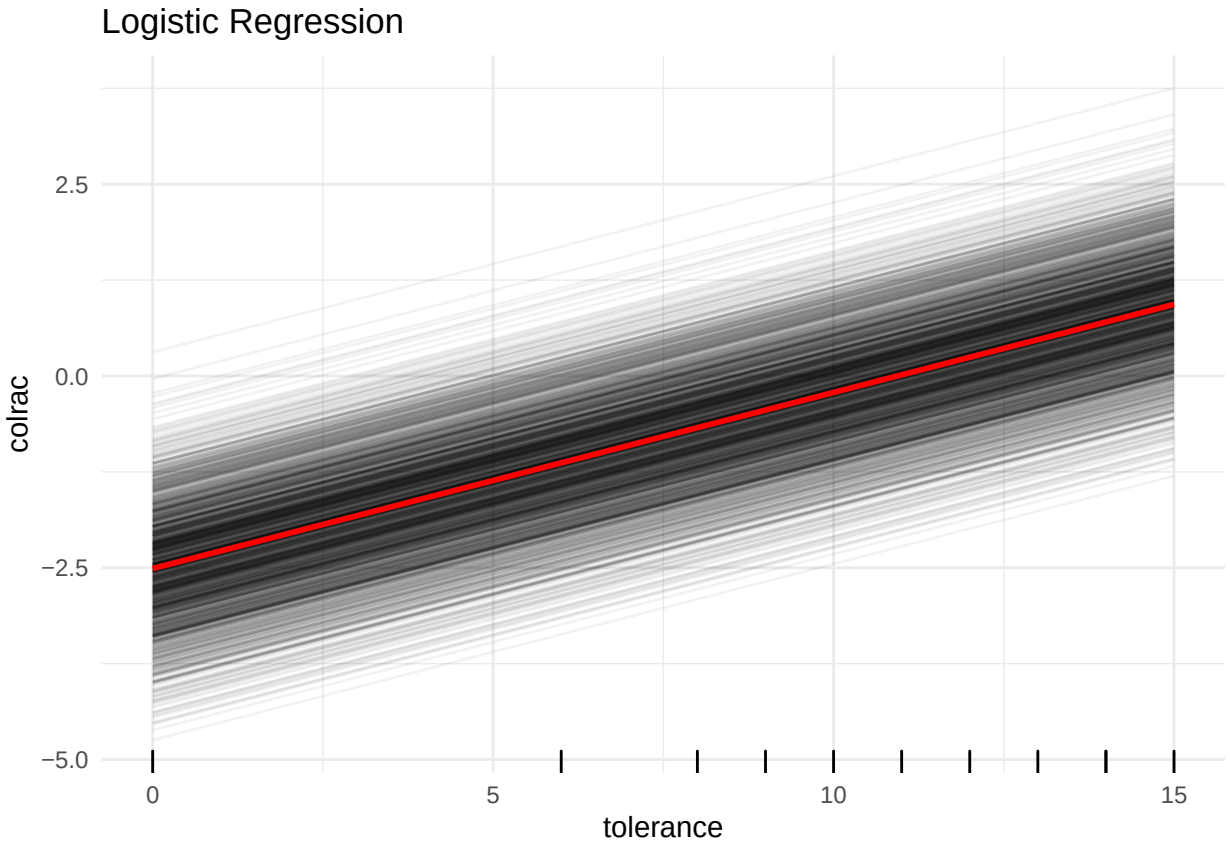
On the other hand, there is a positive trend between `tolerance` and `colrac` from `tolerance`'s PDP/ICE. There doesn't seem to be any outlier regarding to `tolerance` according to the ICE. Again, the shape of the PDP/ICE is not smooth because the model's nature as decision tree. We can also see interesting sub-region divisions in `tolerance`. When there is no actual observation of `tolerance` $= \{1,2,3,4,5\}$, the prediction of `colrac` stay constant which is desirable and is the result of decision tree's nature.

Let's also look at **logistic regression**'s PDP/ICE since it is the best model considering computing cost and interpretability along with model performance according to test session.

```
logit_final %>% partial(pred.var = "age", grid.resolution = 100, ice = TRUE) %>%
    autoplot(rug = TRUE, train = train, alpha = 0.05) + labs(title = "Logistic Regression",
    y = "colrac")
```




```
logit_final %>% partial(pred.var = "tolerance", grid.resolution = 100, ice = TRUE) %>%
    autoplot(rug = TRUE, train = train, alpha = 0.05) + labs(title = "Logistic Regression",
    y = "colrac")
```



The trends in both `age` and `tolerance`'s PDP/ICE are very close to those of Boosting. This comfirms the observation that both boosting and logistic regression perform similarly well in test session. However, logistic regression's PDP/ICE are straight line and smooth, unlike the boosting's. This matches the observation that logistic regression performs worse than boosting in train session. Having a straight line PDP/ICE also matches my reasoning that it is easier to compute as well as interpret logistic regression than to do so with boosting.