Homework 3: Trees

MACS 30100: Perspectives on Computational Modeling
University of Chicago

Yile Chen

For the most part, I collaberated with Jinfei Zhu, Xi Cheng, and Boya Fu. There are changes in Question 3 onwards.

Overview

For each of the following prompts, produce responses *with* code in-line. While you are encouraged to stage and draft your problem set solutions using any files, code, and data you'd like within the private repo for the assignment, *only the final, rendered PDF with responses and code in-line will be graded.*

A Conceptual Problem

1. (15 points) Of the Gini index, classification error, and cross-entropy in simple classification settings with two classes, which would be best to use when *growing* a decision tree? Which would be best to use when *pruning* a decision tree? Why?

Classification error rate is the fraction of the training observations in that region that do not belong to the most common class, which is calculated by: $E = 1 - \max_k(\hat{p}_{mk})$, where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class. However, in practice people tend to opt for Gini Index or Cross-Entropy, because they tend to b

Gini index is a measure of total variance across the K classes, which is defined as: $G = \Sigma_{k=1}^{K}\hat{p}_{mk}(1 - \hat{p}_{mk})$ Gini Index is small if all of $\hat{p}_{mk}$ are close to zero or one. For this reason the Gini index is referred to as a measure of node purity–a samll value indicates that a node contains predominantly observations from a single class.

Cross-entropy is defined as $D = -\Sigma_{k=1}^{K}\hat{p}_{mk}log(\hat{p}_{mk})$. It's similar to Gini index. When more observations are more purely classified, cross-entropy will also shrink towards 0.

When growing a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split, since these two approaches are more senstive to node purity than is the classification error rate.

When pruning the tree, the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal, though any of these three approaches might be used.

An Applied Problem

For the applied portion, your task is to predict attitudes towards racist college professors using the General Social Survey (GSS) survey data. Each respondent was asked *Should a person who believes that Blacks are genetically inferior be allowed to teach in a college or university?* Given the controversy over Richard J. Herrnstein and Charles Murray's *The Bell Curve* and the ostracization of Nobel laureate James Watson over his controversial views on race and intelligence, this applied task will provide additional insight in the public debate over this issue.

To address this problem, use the `gss_*.csv` data sets, which contain a selection of features from the 2012 GSS. The outcome of interest is `colrac`, which is a binary feature coded as either `ALLOWED` or `NOT ALLOWED`, where 1 means the racist professor *should* be allowed to teach, and 0 means the racist professor *should not* be allowed to teach. Full documentation can be found here. I preprocessed the data for you to ease the model-fitting process:

- Missing values have been imputed
- Categorical features with low-frequency classes collapsed into an "other" category
- Nominal features with more than two classes have been converted to dummy features
- Remaining categorical features have been converted to integer values

Your task is to construct a series of models to accurately predict an individual's attitude towards permitting professors who view Blacks to be racially inferior to teach in a college classroom. The learning objectives are:

- Implement a battery of tree-based learners
- Tune hyperparameters
- Substantively interpret models

2. (35 points) Fit the following four tree-based models predicting `colrac` using the training set (`gss_train.csv`) with 10-fold CV. Remember to tune the relevant hyperparameters for each model as necessary. Only use the tuned model with the best performance for the remaining exercises. **Be sure to leave sufficient *time* for hyperparameter tuning, as grid searches can be quite computationally taxing and take a while.**

    - Decision tree (the rpart algorithm)
    - Bagging
    - Random forest
    - Gradient boosting

```
library(tidyverse)
library(tidymodels)
library(caret)
library(pROC)
library(MASS)
library(rcfss)
```

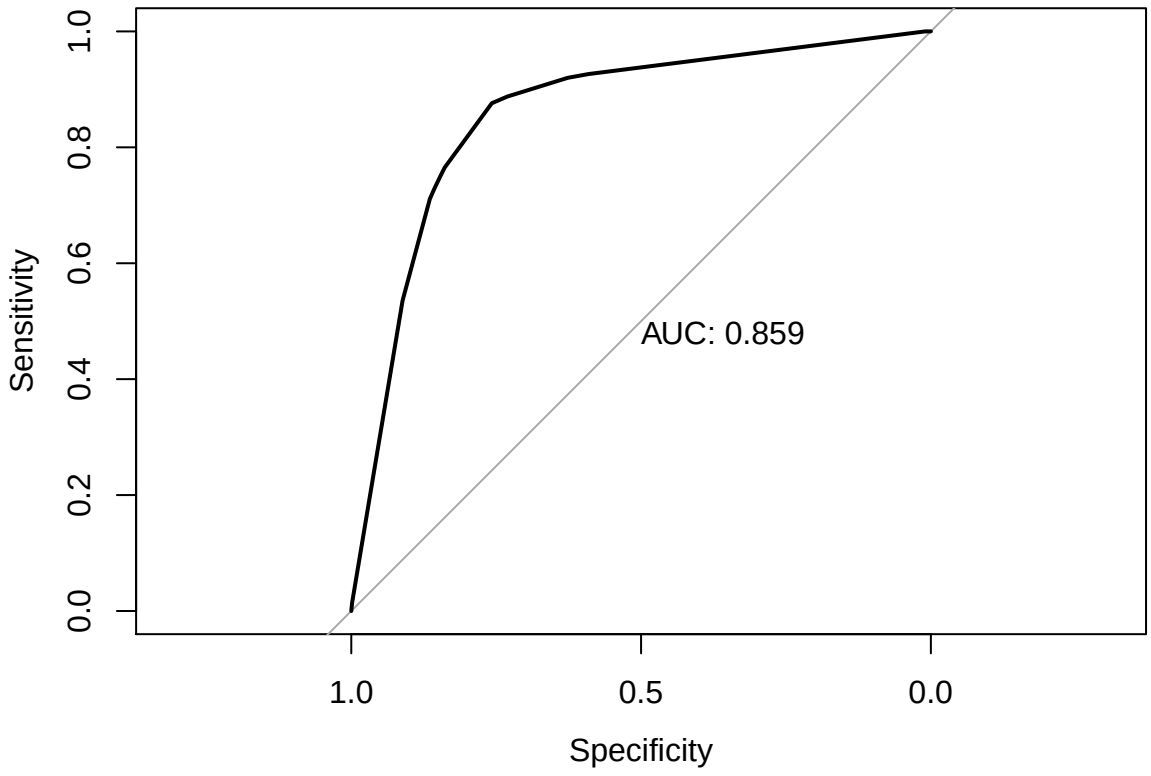
```
gss_test <- read_csv("data/gss_test.csv") %>%
  mutate(colrac = as.factor(colrac))
gss_train <- read_csv("data/gss_train.csv") %>%
  mutate(colrac = as.factor(colrac))

train_x <- gss_train[, setdiff(names(gss_train), "colrac")]
train_y <- gss_train$colrac
test_x <- gss_test[, setdiff(names(gss_train), "colrac")]
test_y <- gss_test$colrac

levels(gss_train$colrac) <- (c("not allowed", "allowed"))
```

```
##RPART##
set.seed(1234)
rpart <- train(colrac ~., gss_train,
               method = "rpart", tuneLength = 10,
               trControl = trainControl(method = "cv", number = 10))
rpart
```

```
## CART
##
```

```
## 1476 samples
##   55 predictor
##    2 classes: 'not allowed', 'allowed'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1328, 1328, 1328, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.003328578  0.7825473  0.5622829
##   0.003566334  0.7825473  0.5622829
##   0.004279601  0.7845882  0.5670028
##   0.005706134  0.7852500  0.5686276
##   0.006419401  0.7791690  0.5565053
##   0.007132668  0.7791690  0.5565053
##   0.008559201  0.7785025  0.5547549
##   0.012838802  0.7730787  0.5431162
##   0.023537803  0.7683352  0.5303206
##   0.499286733  0.6056076  0.1758078
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.005706134.
```

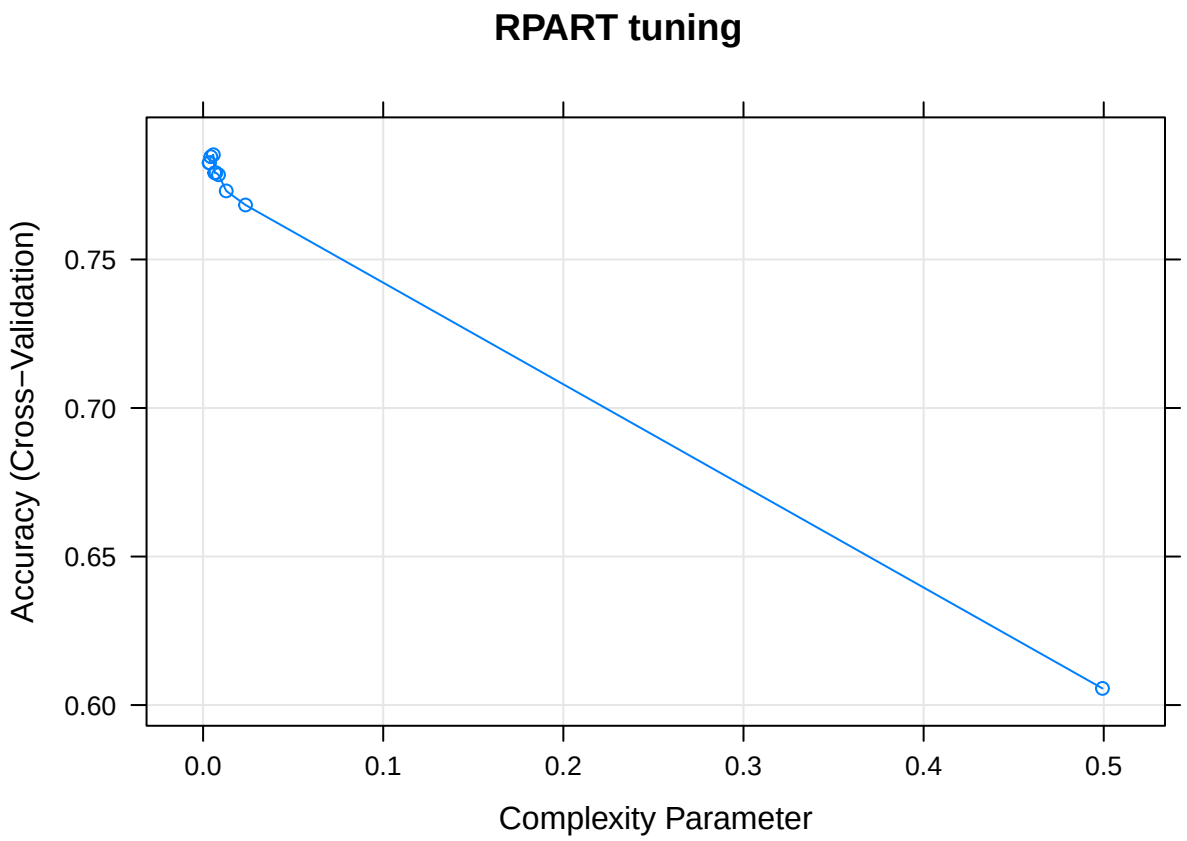
```
rpart_roc <- roc(gss_train$colrac, predict(rpart, train_x, type = "prob")$allowed, plot = TRUE, print.au
```




```
rpart_auc <- rpart_roc$auc

rpart_error <- 1-max(rpart$results$Accuracy)
cat('decision tree error rate', rpart_error)
```

```
## decision tree error rate 0.21475
```

```
plot(rpart, main = "RPART tuning")
```

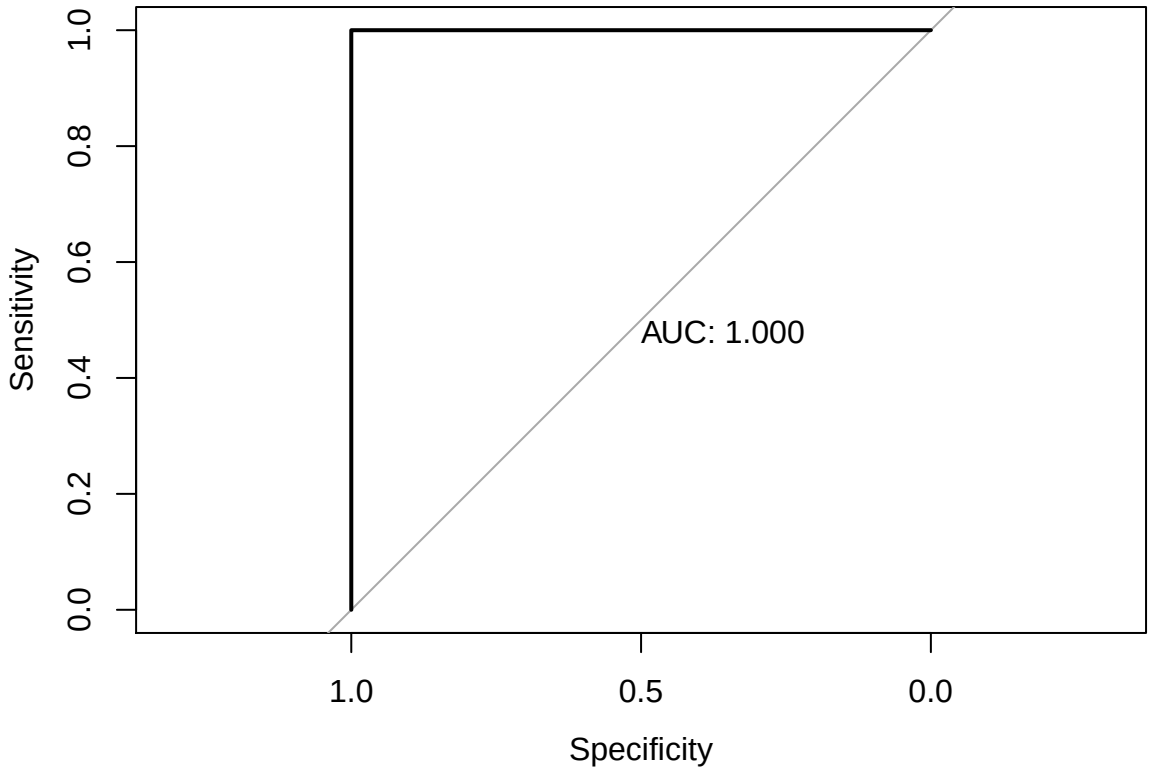



```
##Random Forest##
set.seed(1234)
rf <- train(colrac ~., gss_train,
                method = "rf", tuneLength = 10,
                trControl = trainControl(method = "cv", number = 10))
rf
```

```
## Random Forest
##
## 1476 samples
##   55 predictor
##    2 classes: 'not allowed', 'allowed'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1328, 1328, 1328, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7961528  0.5892971
##    7    0.8076898  0.6123944
##   13    0.8110498  0.6191508
##   19    0.8103558  0.6177666
##   25    0.8103604  0.6179028
```

```
##   31    0.8022477  0.6016817
##   37    0.8063201  0.6099079
##   43    0.7981752  0.5935665
##   49    0.7914231  0.5799965
##   55    0.7968239  0.5908294
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 13.
```

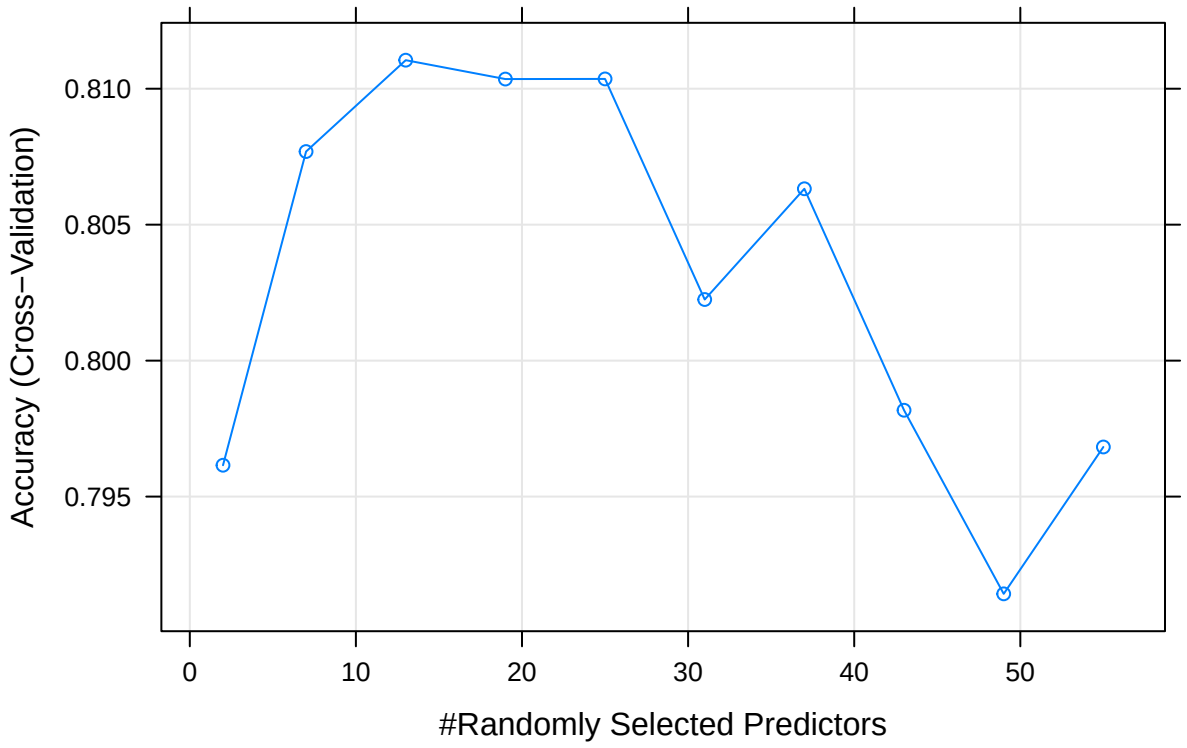
```
rf_roc <- roc(gss_train$colrac, predict(rf, train_x, type = "prob")$allowed, plot = TRUE, print.auc=TRU
```




```
rf_auc <- rf_roc$auc

rf_error <- 1-max(rf$results$Accuracy)
rf_error
```

```
## [1] 0.1889502
```

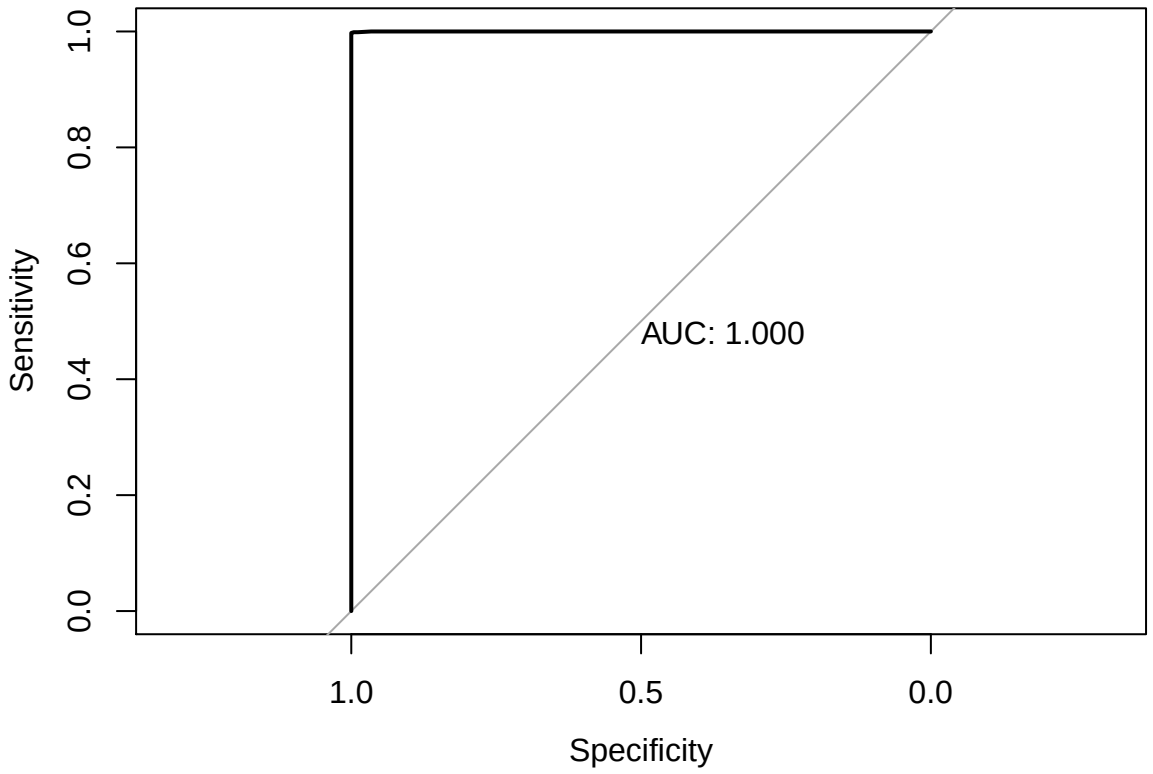
```
plot(rf, main = "Random forest tuning")
```




```
##Bagging##
set.seed(1234)
bagging <- train(colrac ~., gss_train,
                 method = "treebag",
                 trControl = trainControl(method = "cv", number = 10))
bagging
```

```
## Bagged CART
##
## 1476 samples
##   55 predictor
##    2 classes: 'not allowed', 'allowed'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1328, 1328, 1328, 1328, 1329, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7697279  0.5362611
```

```
bagging_roc <- roc(gss_train$colrac, predict(bagging, train_x, type = "prob")$allowed, plot = TRUE, pri
```




```
bagging_auc <- bagging_roc$auc
bagging_auc
```

```
## Area under the curve: 1
```

```
bagging_error <- 1-bagging$results$Accuracy
bagging_error
```

```
## [1] 0.2302721
```

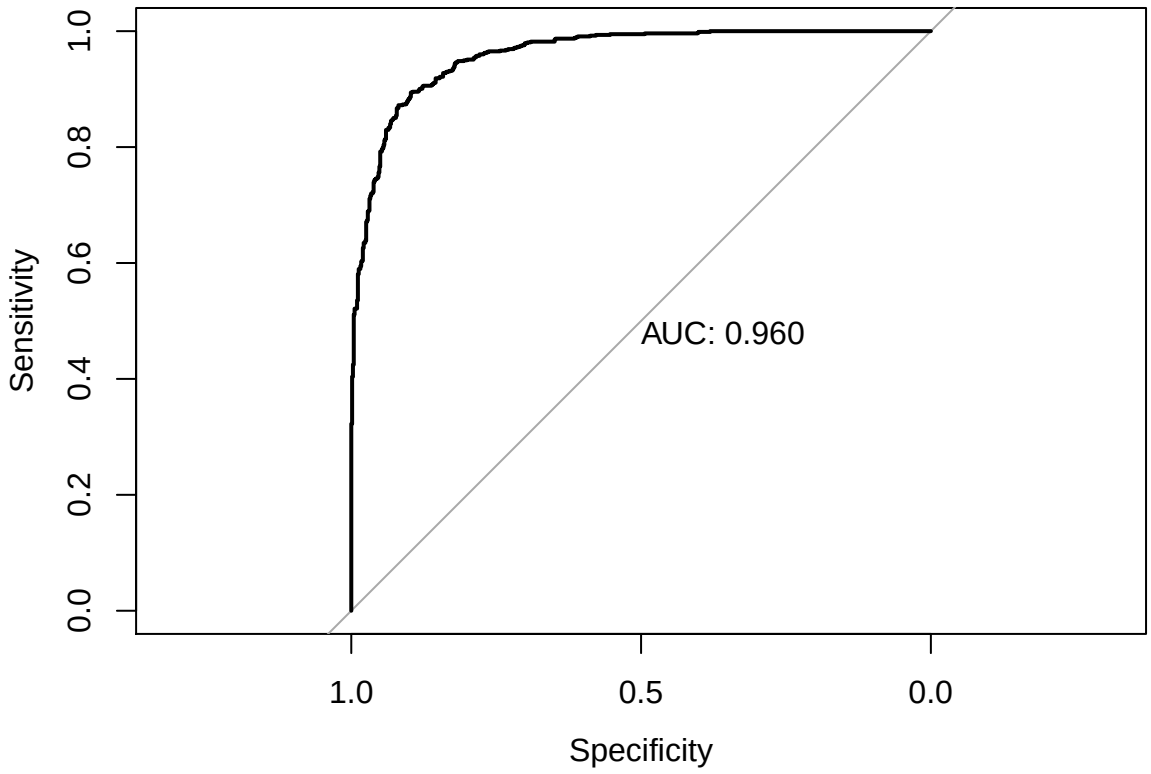
```
##Boosting##
set.seed(1234)
gbm <- train(colrac ~., gss_train,
               method = "gbm", tuneLength = 10,
               trControl = trainControl(method = "cv", number = 10), verbose = FALSE)

gbm
```

```
## Stochastic Gradient Boosting
##
## 1476 samples
##   55 predictor
##    2 classes: 'not allowed', 'allowed'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1328, 1328, 1328, 1328, 1328, 1329, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##    1                  50      0.7846387  0.5644528
```

```
##    1                  100       0.7988647  0.5941425
##    1                  150       0.8049641  0.6065587
##    1                  200       0.8015674  0.5998945
##    1                  250       0.8042701  0.6058596
##    1                  300       0.8076577  0.6126425
##    1                  350       0.8042517  0.6060021
##    1                  400       0.8049320  0.6075090
##    1                  450       0.8089722  0.6157298
##    1                  500       0.8056122  0.6088450
##    2                   50       0.7961620  0.5885290
##    2                  100       0.8002068  0.5972691
##    2                  150       0.8022568  0.6017763
##    2                  200       0.7995266  0.5968000
##    2                  250       0.7913909  0.5805572
##    2                  300       0.7920757  0.5820097
##    2                  350       0.7913817  0.5805089
##    2                  400       0.7920666  0.5818629
##    2                  450       0.7893501  0.5768134
##    2                  500       0.7859533  0.5699492
##    3                   50       0.7988417  0.5943076
##    3                  100       0.8069774  0.6112633
##    3                  150       0.8042747  0.6062996
##    3                  200       0.8015306  0.6006827
##    3                  250       0.8015306  0.6010477
##    3                  300       0.8029003  0.6035914
##    3                  350       0.8042425  0.6065988
##    3                  400       0.8069544  0.6118712
##    3                  450       0.8015398  0.6012980
##    3                  500       0.8069498  0.6118613
##    4                   50       0.8042333  0.6056946
##    4                  100       0.8035576  0.6043773
##    4                  150       0.7974720  0.5923413
##    4                  200       0.8022155  0.6018237
##    4                  250       0.7995082  0.5966077
##    4                  300       0.7988417  0.5955123
##    4                  350       0.7934133  0.5846480
##    4                  400       0.7961114  0.5901171
##    4                  450       0.7954357  0.5889162
##    4                  500       0.7961068  0.5902662
##    5                   50       0.8069958  0.6110261
##    5                  100       0.8103466  0.6179838
##    5                  150       0.8076163  0.6129899
##    5                  200       0.8055847  0.6088806
##    5                  250       0.8062649  0.6101549
##    5                  300       0.8082690  0.6143080
##    5                  350       0.8055709  0.6091884
##    5                  400       0.8035622  0.6048069
##    5                  450       0.8055801  0.6088577
##    5                  500       0.8082874  0.6145419
##    6                   50       0.8022614  0.6018194
##    6                  100       0.8076531  0.6125810
##    6                  150       0.8022477  0.6020822
##    6                  200       0.7981430  0.5937042
##    6                  250       0.7913909  0.5800251
```

```
##    6                  300      0.7934317  0.5839645
##    6                  350      0.7940890  0.5856095
##    6                  400      0.8008779  0.5994354
##    6                  450      0.7981660  0.5938560
##    6                  500      0.7934225  0.5844034
##    7                   50      0.8096847  0.6167975
##    7                  100      0.8056168  0.6084528
##    7                  150      0.8069682  0.6114828
##    7                  200      0.8008917  0.5990370
##    7                  250      0.7927468  0.5828130
##    7                  300      0.7893409  0.5758662
##    7                  350      0.7920712  0.5814624
##    7                  400      0.7907060  0.5788160
##    7                  450      0.7947830  0.5869196
##    7                  500      0.7947739  0.5870480
##    8                   50      0.7968055  0.5910153
##    8                  100      0.7913679  0.5801746
##    8                  150      0.7947739  0.5868814
##    8                  200      0.7961206  0.5896631
##    8                  250      0.7968239  0.5911748
##    8                  300      0.7981522  0.5938838
##    8                  350      0.7954633  0.5884159
##    8                  400      0.7934409  0.5843907
##    8                  450      0.7934409  0.5844398
##    8                  500      0.7981752  0.5940572
##    9                   50      0.7974949  0.5921684
##    9                  100      0.8049228  0.6074047
##    9                  150      0.7954449  0.5885776
##    9                  200      0.7988187  0.5954849
##    9                  250      0.7981384  0.5941004
##    9                  300      0.7933811  0.5847139
##    9                  350      0.8001701  0.5983124
##    9                  400      0.8028820  0.6034518
##    9                  450      0.7954357  0.5886207
##    9                  500      0.7988279  0.5954146
##   10                   50      0.8076163  0.6128865
##   10                  100      0.7988095  0.5951581
##   10                  150      0.7968055  0.5911195
##   10                  200      0.7961298  0.5896508
##   10                  250      0.8042425  0.6061133
##   10                  300      0.7954633  0.5885244
##   10                  350      0.7988463  0.5952500
##   10                  400      0.8002022  0.5979582
##   10                  450      0.8001931  0.5980430
##   10                  500      0.8029187  0.6035578
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## 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 = 10.
```

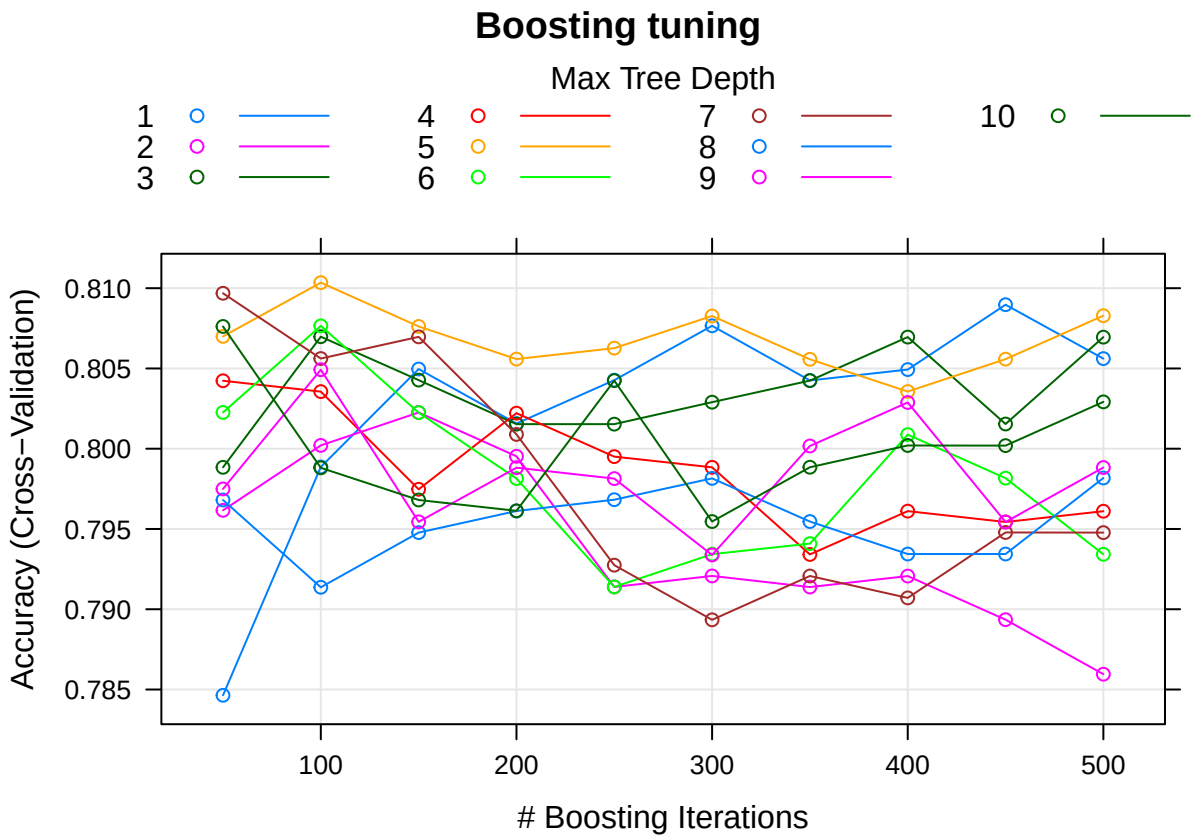
```
gbm_roc <- roc(gss_train$colrac, predict(gbm, train_x, type = "prob")$allowed, plot = TRUE, print.auc=TR
```




```
gbm_auc <- gbm_roc$auc


gbm_error <- 1-max(gbm$results$Accuracy)
gbm_error
```

```
## [1] 0.1896534
```

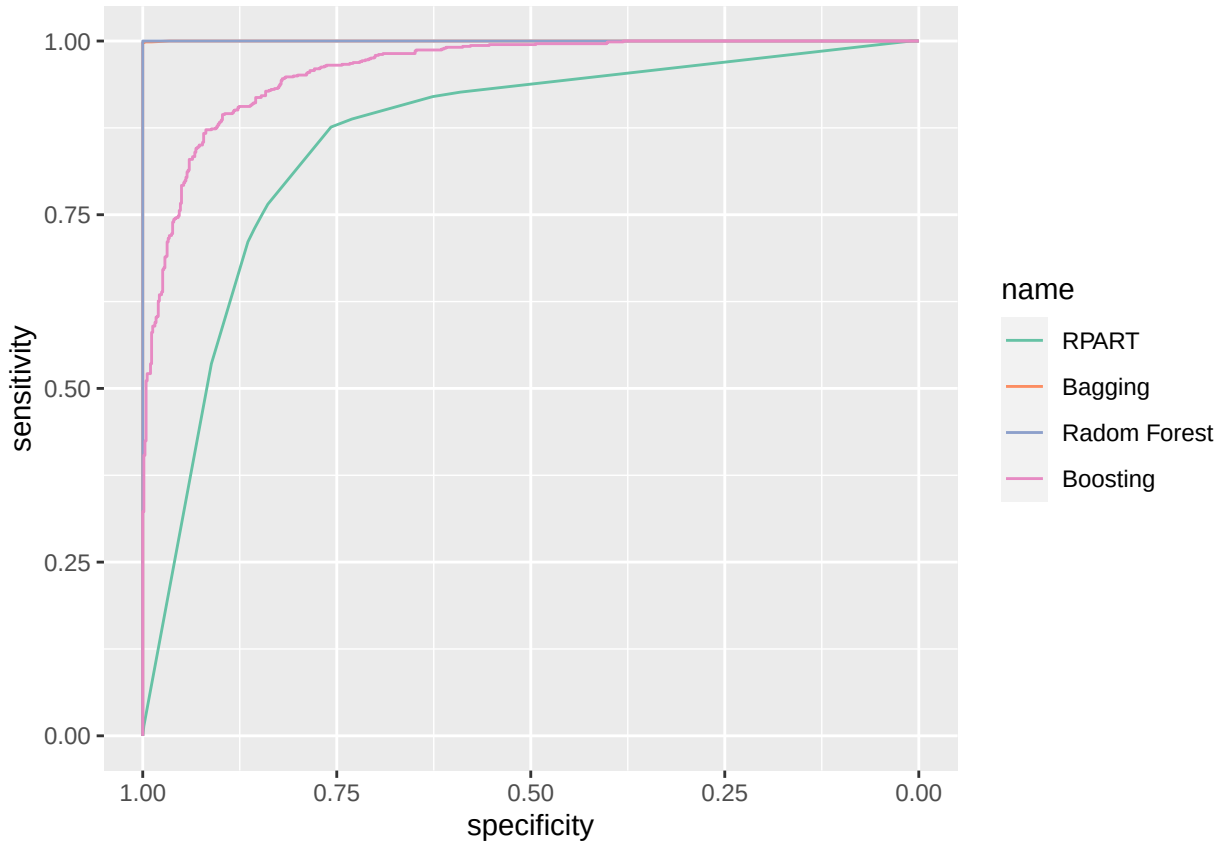
```
plot(gbm, main = "Boosting tuning")
```



3. (20 points) Compare and present each model's (training) performance based on:

* Cross-validated error rate
* ROC/AUC

```
ggroc(list(rpart_roc, bagging_roc, rf_roc, gbm_roc)) +
   scale_color_brewer(palette = "Set2", labels = c("RPART","Bagging","Radom Forest","Boosting"))
```




```
tibble(models = c("RPART","Bagging","Random Forest","Boosting"), auc = c(rpart_auc, bagging_auc, rf_auc
```

```
## # A tibble: 4 x 3
##   models          auc error
##   <chr>         <dbl> <dbl>
## 1 RPART         0.859 0.215
## 2 Bagging       1.00  0.230
## 3 Random Forest 1     0.189
## 4 Boosting      0.960 0.190
```

ROC/AUC of these 4 models and cross-validated error rate are in the graph and dataframe.

4. (15 points) Which is the best model? Defend your choice.

As displayed above, I find that bagging and random forest performs the best according to ROC/AUC (the difference in their AUC is less than 0.0001). With a ROC/AUC nearly equals to 1, it means the two models can make predictions highly accurately on the training data (false positive rate = 0 and true positive rate = 1). Meanwhile, the random forest model also has the lowest cross-validated error rate. So according to the two metrics, I think that random forest is the best model. Nevertheless, with such high ROC/AUC value, we should also be cautious that the model may overfit the data.

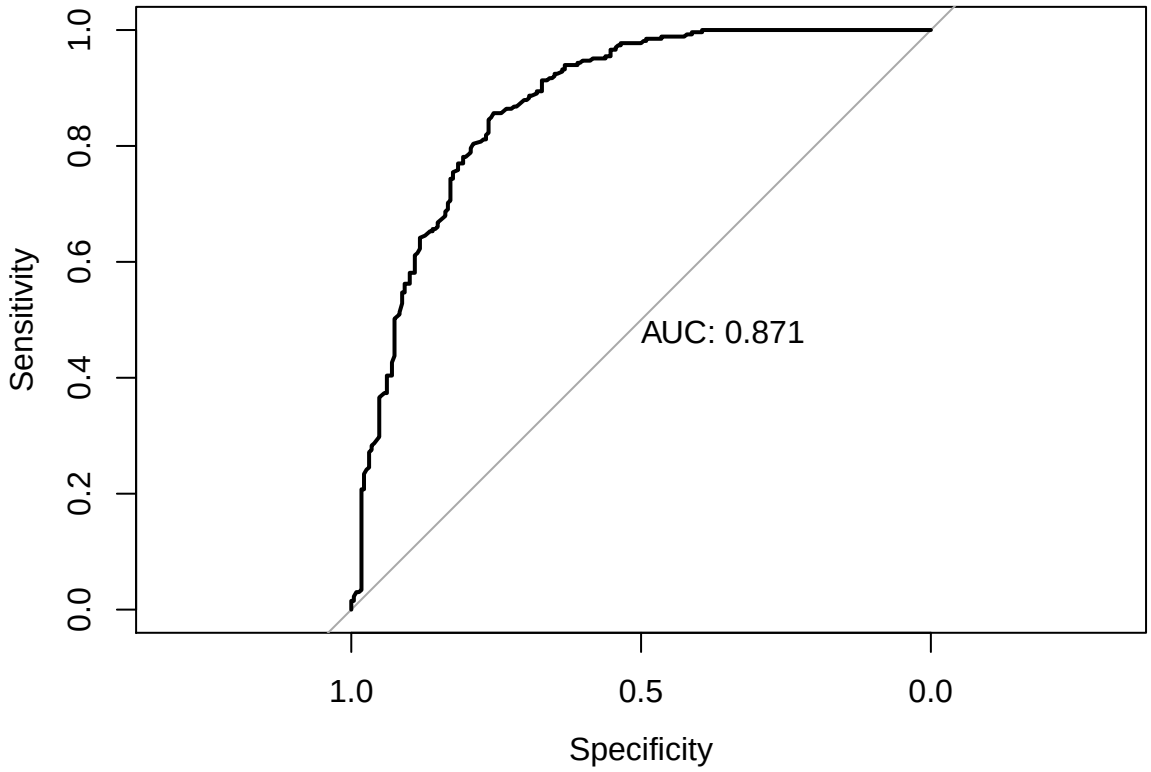
5. (15 points) Evaluate the performance of the best model selected in the previous question using the test set (`gss_test.csv`) by calculating and presenting the classification error rate and AUC of this model. Compared to the fit evaluated on the training set, does this "best" model generalize well? Why or why not? How do you know?

```
#calculate error rate
rf_predict_test <- predict(rf, test_x)
levels(gss_test$colrac) <- (c("not allowed", "allowed"))
rf_error_test <- mean(rf_predict_test != gss_test$colrac)
```

```r
print(rf_error_test)
```

```
## [1] 0.2068966
```

```r
# calculate AUC
rf_test_roc <- roc(gss_test$colrac, predict(rf, test_x, type = "prob")$allowed, plot = TRUE, print.auc="
```




```r
rf_test_auc <- rf_test_roc$auc
confusionMatrix(gss_test$colrac, predict(rf, test_x))
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    not allowed allowed
##   not allowed         155      73
##   allowed              28     237
##
##                Accuracy : 0.7951
##                  95% CI : (0.7568, 0.8299)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : 1.023e-15
##
##                   Kappa : 0.5822
##
##  Mcnemar's Test P-Value : 1.197e-05
##
##             Sensitivity : 0.8470
##             Specificity : 0.7645
##          Pos Pred Value : 0.6798
##          Neg Pred Value : 0.8943
```

```
##              Prevalence : 0.3712
##          Detection Rate : 0.3144
##    Detection Prevalence : 0.4625
##       Balanced Accuracy : 0.8058
##
##        'Positive' Class : not allowed
##
```

```
print(rf_test_auc)
```

```
## Area under the curve: 0.8715
```

The test error rate of the tuned random forest model is 0.2048, which is higher than its cross-validated training error rate 0.1889502. The AUC of the test set is 0.871, which is lower than the training AUC of 1. These results suggest this 'best' model is acceptable, but it does not generalize terribly well. There may be slight over-fitting: the random forest model fits the training data too well and could have pruned branches more effectively.