Problem Set 5

Minyoung Do

3/1/2020

Conceptual: Cost functions for classification trees

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

The optimal cost function for each modeling procedure is: cross-entropy for grorwing a tree, and classification error rate for pruing the tree. Cross-entropy is always larger than the averaged Entropy because of its "bell shape," which is why it keeps splitting the nodes while the classification error does not. Therefore, cross-entropy is more sensitive to the purity of nodes, thus evaluating the quality of splits better. I would like to note that the classification error rate does not always underperform compared to other two methods; it is preferred when the final goal is to enhance the prediction accuracy of the final pruned tree. Classification error uses the most commonly occurring class in a region to make splits as we learned in class. Since classification error rate is the best metric for measuring accuracy, it might be a reasonable choice to use classification error rate for growing a pruing the tree. However, considering the decision tree algorithm is *greedy*, an attempt to maximize the accuracy at each split (or in a region) may not end up selecting the *globally* more accurate classifer.

Application: Predicting attitudes towards racist college professors

In this problem set, you are going to predict attitudes towards racist college professors, using the GSS survey data. Specifically, 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 kerfuffle 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 analysis will provide further insight into the public debate over this issue.

gss_*.csv contain a selection of features from the 2012 GSS. The outcome of interest colrac is a binary variable coded as either ALLOWED or NOT ALLOWED, where 1 = the racist professor should be allowed to teach, and 0 = the racist professor should not be allowed to teach. Documentation for the other predictors (if the variable is not clearly coded) can be viewed here. Some data pre-processing has been done in advance for you to ease your model fitting: (1) Missing values have been imputed; (2) Categorical variables with low-frequency classes had those classes collapsed into an "other" category; (3) Nominal variables with more than two classes have been converted to dummy variables; and (4) Remaining categorical variables have been converted to integer values, stripping their original labels

Your mission is to bring trees into the context of other classification approaches, thereby constructing a series of models to accurately predict an individual's attitude towards permitting racist professors to teach. The learning objectives of this exercise are:

```
1. Implement a battery of learners (including trees)
2. Tune hyperparameters
3. Substantively evaluate models
```

Estimate the models

```
gss_test <- read_csv("gss_test.csv")
gss_train <- read_csv("gss_train.csv")
```

(35 points; 5 points/model) Estimate the following models, predicting colrac using the training set (the training .csv) with 10-fold CV:

```
# saving 10-fold CV option as an object for simplifying the codes below.
train_control <- trainControl("cv", 10)

# factor column
gss_train <- gss_train %>%
  mutate(colrac = as.factor(colrac))
gss_test <- gss_test %>%
  mutate(colrac = as.factor(colrac))
```

Logistic regression

```
# function to generate assessment statistics
holdout_res <- function(splits) {
  # Fit the model to the training set
  mod <- glm(colrac ~ ., data = analysis(splits),
             family = binomial)

  # `augment` will save the predictions with the holdout data set
  res <- augment(mod, newdata = assessment(splits)) %>%
    as_tibble() %>%
    mutate(.prob = logit2prob(.fitted),
           .pred = round(.prob))

  # Return the assessment data set with the additional columns
  res
}

colrac_cv10 <- vfold_cv(data = gss_train,
                        v = 10) %>%
  mutate(results = map(splits, holdout_res)) %>%
  unnest(results) %>%
  mutate(.pred = factor(.pred)) %>%
  group_by(id) %>%
  accuracy(truth = colrac, estimate = .pred)
```

```r
# error rate
logit_mse <- 1 - mean(colrac_cv10$.estimate, na.rm = TRUE)
```

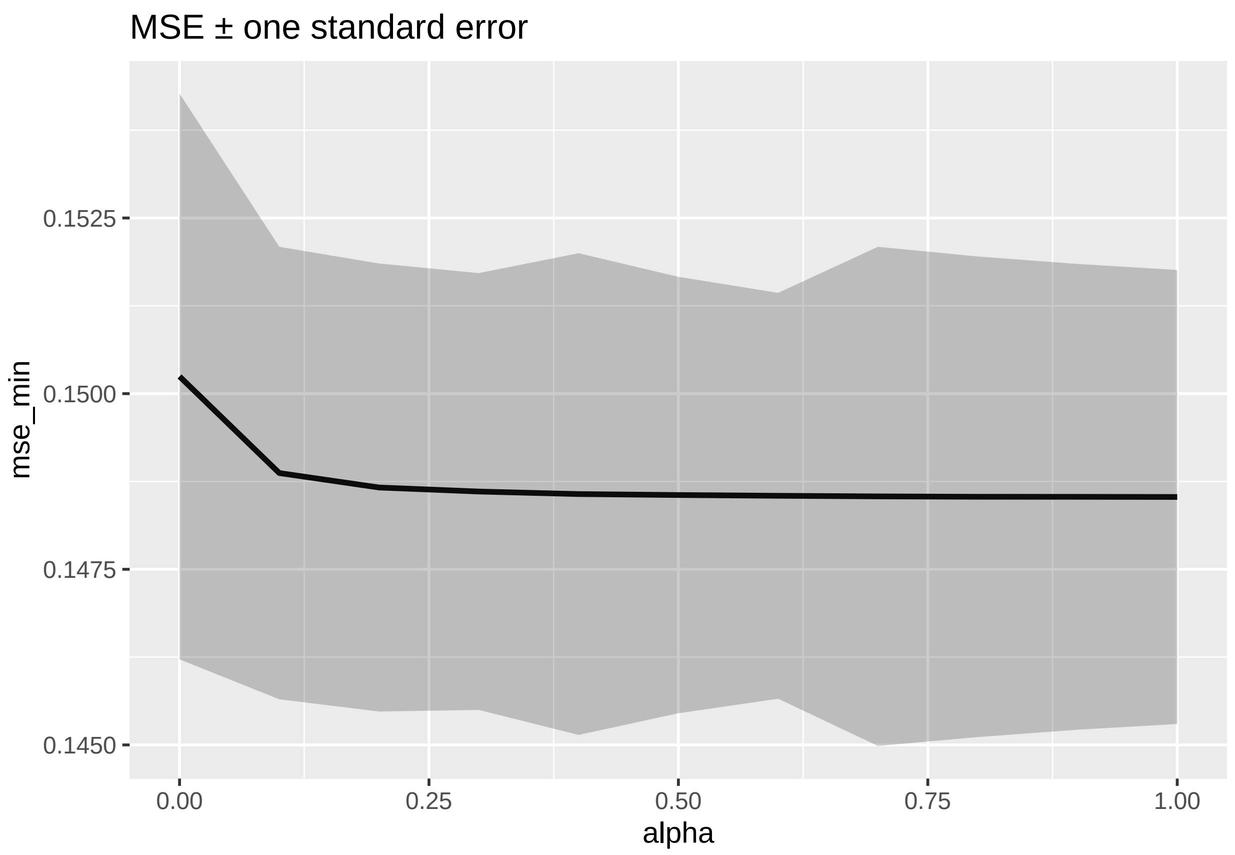
Naive Bayes

```r
gss_train_nb <- gss_train %>%
  mutate(colrac = ifelse(colrac == 1, "TRUE", "FALSE"))

search_grid <- expand.grid(
  usekernel = TRUE,
  fL = 1,
  adjust = seq(0, 5, by = 1)
)

# train model
nb_gss <- suppressWarnings(
  train(colrac ~ .,
        data = gss_train_nb,
        method = "nb",
        trControl = train_control,
        tuneGrid = search_grid))
nb_gss
```

```
## Naive Bayes
##
## 1481 samples
##   77 predictor
##    2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1334, 1333, 1332, 1332, 1333, ...
## Resampling results across tuning parameters:
##
##   adjust  Accuracy   Kappa
##   0             NaN        NaN
##   1       0.7001996  0.4085824
##   2       0.6961272  0.4017685
##   3       0.6860192  0.3832523
##   4       0.6806229  0.3738679
##   5       0.6657578  0.3467963
##
## Tuning parameter 'fL' was held constant at a value of 1
## Tuning
##  parameter 'usekernel' was held constant at a value of TRUE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 1, usekernel = TRUE
##  and adjust = 1.
```

Elastic net regression

```r
gss_train_x <- model.matrix(colrac ~ ., gss_train)[, -1]
gss_train_y <- gss_train$colrac %>%
  as.numeric()

lasso    <- glmnet(gss_train_x, gss_train_y, alpha = 1.0)
elastic1 <- glmnet(gss_train_x, gss_train_y, alpha = 0.25)
elastic2 <- glmnet(gss_train_x, gss_train_y, alpha = 0.75)
ridge    <- glmnet(gss_train_x, gss_train_y, alpha = 0.0)

# grid search for varying alpha
fold_id <- sample(1:10, size = length(gss_train_y), replace = TRUE) # maintain the same folds

# search across a range of alphas
tuning_grid <- tibble(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,
  mse_1se    = NA,
  lambda_min = NA,
  lambda_1se = NA
)

for(i in seq_along(tuning_grid$alpha)) {
  # fit CV model for each alpha value
  fit <- cv.glmnet(gss_train_x,
                   gss_train_y,
                   alpha = tuning_grid$alpha[i],
                   foldid = fold_id)

  # extract MSE and lambda values
  tuning_grid$mse_min[i]    <- fit$cvm[fit$lambda == fit$lambda.min]
  tuning_grid$mse_1se[i]    <- fit$cvm[fit$lambda == fit$lambda.1se]
  tuning_grid$lambda_min[i] <- fit$lambda.min
  tuning_grid$lambda_1se[i] <- fit$lambda.1se
}

tuning_grid %>%
  mutate(se = mse_1se - mse_min) %>%
  ggplot(aes(alpha, mse_min)) +
  geom_line(size = 1) +
  geom_ribbon(aes(ymax = mse_min + se, ymin = mse_min - se), alpha = .25) +
  ggtitle("MSE ± one standard error")
```



```r
# minimum MSE
min(tuning_grid[,2])
```

```
## [1] 0.1485292
```

```r
# combination of alpha = 1 & lambda = 0.1488064
coef_en <- coef(fit, alpha = 1)
```

Decision tree (CART)

```r
gss_cart <- train(
  x = as.matrix(gss_train[, -8]),
  y = gss_train$colrac,
  method = "rpart",
  trControl = train_control
)
gss_cart
```

```
## CART
##
## 1481 samples
##   77 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1334, 1332, 1333, 1333, 1334, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01422475  0.7764786  0.5493188
##   0.02275960  0.7588513  0.5098989
##   0.50071124  0.6070216  0.1790943
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01422475.
```

1 - 0.7757985

Bagging

```
gss_bag <- train(
  x = as.matrix(gss_train[, -8]),
  y = gss_train$colrac,
  method = "treebag",
  trControl = train_control
)
gss_bag
```

```
## Bagged CART
##
## 1481 samples
##   77 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1332, 1334, 1333, 1333, 1333, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.786698  0.5701281
```

1 - 0.7859106

Random forest

```r
tune_grid <- expand.grid(.mtry = (1:15))

gss_rf <- train(colrac~.,
                    data = gss_train,
                    method = "rf",
                    tuneGrid = tune_grid,
                    metric = "Accuracy",
                    trControl= train_control)
gss_rf
```

```
## Random Forest
##
## 1481 samples
##   77 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1333, 1332, 1333, 1333, 1332, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7441257  0.4818569
##    2    0.7894146  0.5756903
##    3    0.7954867  0.5880341
##    4    0.8022299  0.6013534
##    5    0.8028874  0.6026787
##    6    0.8049418  0.6069204
##    7    0.7995226  0.5957459
##    8    0.7995317  0.5958606
##    9    0.8002074  0.5969823
##   10    0.7988515  0.5943630
##   11    0.8109955  0.6189153
##   12    0.8042479  0.6051023
##   13    0.8076353  0.6120471
##   14    0.8049190  0.6065589
##   15    0.7995089  0.5959047
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 11.
```

Boosting

```r
lambda_seq <- seq(0.0001, 0.04, by = 0.01)
boost_gss <- gbm(colrac ~ .,
                    data = gss_train,
                    distribution = "gaussian",
                    n.trees = 1000,
                    shrinkage = lambda_seq,
                    interaction.depth = 3,
                 n.minobsinnode = 10)
boost_gss
```

```
## gbm(formula = colrac ~ ., distribution = "gaussian", data = gss_train,
##     n.trees = 1000, interaction.depth = 3, n.minobsinnode = 10,
##     shrinkage = lambda_seq)
## A gradient boosted model with gaussian loss function.
## 1000 iterations were performed.
## There were 77 predictors of which 20 had non-zero influence.
```

Evaluate the models

(20 points) Compare and present each model's (training) performance

Cross-validated error rate

1. Logistic Regression: 0.2086
2. Naive Bayes: 0.2594
3. Elastic Net: 0.1488
4. Decision Tree (CART): 0.2242
5. Bagging: 0.2140
6. Random Forest: 0.1944
7. Boosting: 0.2103

ROC/AUC

1. Logistic Regression: 0.8143
2. Naive Bayes: 0.8393
3. Elastic Net: 0.8955
4. Decision Tree (CART): 0.7757
5. Bagging: 0.7859
6. Random Forest: 0.8914
7. Boosting: 0.7897

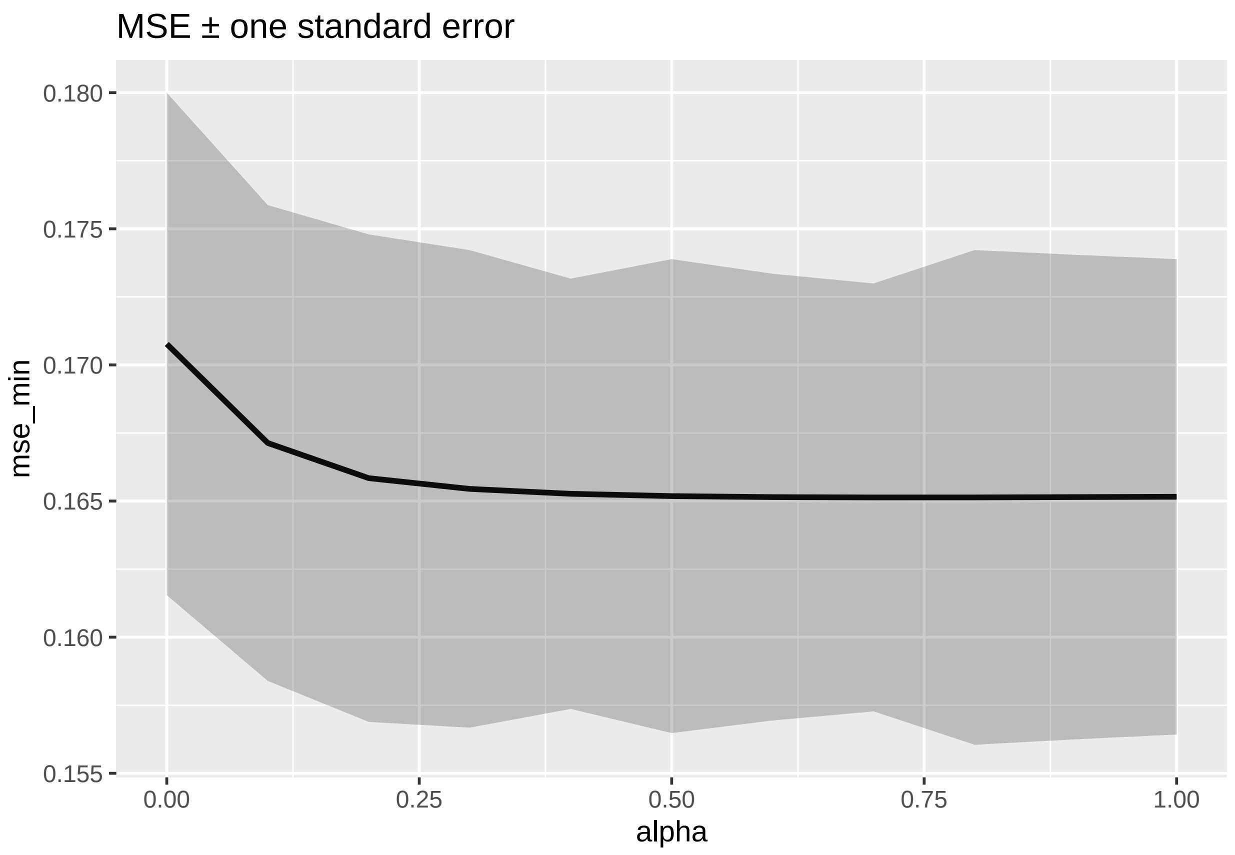
(15 points) Which is the best model? Defend your choice.

Of all 7 models, elastic net regression yields the minimum error rate and higher AUC, which makes it rather obvious that it is the best model. The AUC results indicate the prediction accuracy of elastic net and random forest is very similar, by 0.004; however, since elastic net regression has the minimum error rate, we can conclude that the best model for this data is elastic net regression. It is important to note that, as elastic net has the minimum error rate when $\alpha = 1$, the best model for our data is **LASSO** regression.

Evaluate the best model

(15 points) Evaluate the final, best model's (selected in 4) performance on the test set (the test .csv) by calculating and presenting the classification error rate and AUC. Compared to the fit evaluated on the training set in questions 3-4, does the "best" model generalize well? Why or why not? How do you know?

```r
gss_test_x <- model.matrix(colrac ~ ., gss_test)[, -1]
gss_test_y <- gss_test$colrac %>%
  as.numeric()

lasso    <- glmnet(gss_test_x, gss_test_y, alpha = 1.0)
elastic1 <- glmnet(gss_test_x, gss_test_y, alpha = 0.25)
elastic2 <- glmnet(gss_test_x, gss_test_y, alpha = 0.75)
ridge    <- glmnet(gss_test_x, gss_test_y, alpha = 0.0)

# grid search for varying alpha
fold_id <- sample(1:10, size = length(gss_test_y), replace = TRUE) # maintain the same folds a

# search across a range of alphas
tuning_grid2 <- tibble(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,
  mse_1se    = NA,
  lambda_min = NA,
  lambda_1se = NA
)

for(i in seq_along(tuning_grid2$alpha)) {
  # fit CV model for each alpha value
  fit2 <- cv.glmnet(gss_test_x,
                   gss_test_y,
                   alpha = tuning_grid2$alpha[i],
                   foldid = fold_id)

  # extract MSE and lambda values
  tuning_grid2$mse_min[i]    <- fit2$cvm[fit2$lambda == fit2$lambda.min]
  tuning_grid2$mse_1se[i]    <- fit2$cvm[fit2$lambda == fit2$lambda.1se]
  tuning_grid2$lambda_min[i] <- fit2$lambda.min
  tuning_grid2$lambda_1se[i] <- fit2$lambda.1se
}

tuning_grid2 %>%
  mutate(se = mse_1se - mse_min) %>%
  ggplot(aes(alpha, mse_min)) +
  geom_line(size = 1) +
  geom_ribbon(aes(ymax = mse_min + se, ymin = mse_min - se), alpha = .25) +
  ggtitle("MSE ± one standard error")
```



```r
# minimum MSE
min(tuning_grid2[,2])
```

```
## [1] 0.1651323
```

```r
# combination of alpha = 1 & lambda = 0.1488064
coef_en <- coef(fit2, alpha = 1)

  fit2 <- cv.glmnet(gss_test_x,
                    gss_test_y,
                    alpha = tuning_grid2$alpha[i],
                    foldid = fold_id)
```

The test error of elastic net regression is 0.1669 when $\alpha = 0.4$, which is higher than the training error (0.1488064) only by 0.01. Even when performing elastic net regression on both sets at $\alpha = 1$, the difference remains small. This result makes sense as the training error usually is almost always higher, unless the training data is not properly splitted. Therefore, I conclude that this elastic net regression model generalizes pretty well.