STA 5936: Homework 1

Due: Friday, January 16th by 11:59 PM

Use a programming language or package where decision trees and random forests can be trained and applied. Examples include Matlab, Python (scikit-learn package), or R.

# Load in packages  
library(tidymodels)

## Warning: package 'tidymodels' was built under R version 4.4.2

## ── Attaching packages ────────────────────────────────────── tidymodels 1.2.0 ──

## ✔ broom 1.0.7 ✔ recipes 1.1.0  
## ✔ dials 1.3.0 ✔ rsample 1.2.1  
## ✔ dplyr 1.1.4 ✔ tibble 3.2.1  
## ✔ ggplot2 3.5.1 ✔ tidyr 1.3.1  
## ✔ infer 1.0.7 ✔ tune 1.2.1  
## ✔ modeldata 1.4.0 ✔ workflows 1.1.4  
## ✔ parsnip 1.2.1 ✔ workflowsets 1.1.0  
## ✔ purrr 1.0.2 ✔ yardstick 1.3.1

## Warning: package 'broom' was built under R version 4.4.2

## Warning: package 'dials' was built under R version 4.4.2

## Warning: package 'scales' was built under R version 4.4.2

## Warning: package 'dplyr' was built under R version 4.4.2

## Warning: package 'ggplot2' was built under R version 4.4.2

## Warning: package 'infer' was built under R version 4.4.2

## Warning: package 'modeldata' was built under R version 4.4.2

## Warning: package 'parsnip' was built under R version 4.4.2

## Warning: package 'purrr' was built under R version 4.4.2

## Warning: package 'recipes' was built under R version 4.4.2

## Warning: package 'rsample' was built under R version 4.4.2

## Warning: package 'tibble' was built under R version 4.4.2

## Warning: package 'tidyr' was built under R version 4.4.2

## Warning: package 'tune' was built under R version 4.4.2

## Warning: package 'workflows' was built under R version 4.4.2

## Warning: package 'workflowsets' was built under R version 4.4.2

## Warning: package 'yardstick' was built under R version 4.4.2

## ── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
## ✖ purrr::discard() masks scales::discard()  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ✖ recipes::step() masks stats::step()  
## • Use suppressPackageStartupMessages() to eliminate package startup messages

library(tidyr)

# Load in the SAT datasets  
SATXTest <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/satimage/Xtest.dat", header = FALSE, sep = " ")  
  
SATX <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/satimage/X.dat", header = FALSE, sep = " ")  
  
SATYTest <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/satimage/Ytest.dat", header = FALSE, sep = " ")  
  
SATY <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/satimage/Y.dat", header = FALSE, sep = " ")  
  
SATTrain <- bind\_cols(as\_tibble(SATX), class = as.factor(SATY$V1))  
  
SATTest <- bind\_cols(as\_tibble(SATXTest), class = as.factor(SATYTest$V1))

# Load in the MAD datasets  
MADTest <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/MADELON/madelon\_test.data", header = FALSE, sep = " ")  
  
MADTrain <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/MADELON/madelon\_train.data", header = FALSE, sep = " ")  
  
MADTrainLabels <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/MADELON/madelon\_train.labels", header = FALSE, sep = " ")  
  
MADValid <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/MADELON/madelon\_valid.data", header = FALSE, sep = " ")  
  
MADValidLabels <- read.csv("C:/Users/Bryce/Downloads/STA 5936/HW1/MADELON/madelon\_valid.labels", header = FALSE, sep = " ")  
  
MADTrain <- bind\_cols(as\_tibble(MADTrain), class = as.factor(MADTrainLabels$V1)) %>% select (-c(V501))  
  
MADValid <- bind\_cols(as\_tibble(MADValid), class = as.factor(MADValidLabels$V1)) %>% select (-c(V501))

Using the training and test sets specified in the syllabus, perform the following tasks:

**a)** On the madelon dataset, train decision trees of maximum depth 1, 2, …. up to 12, for a total of 12 decision trees. If your package does not allow the max depth as a parameter, train trees with 21, 22, …, 212 nodes, again a total of 12 trees. Use the trained trees to predict the class labels on the training and test sets, and obtain the training and test misclassification errors. Plot on the same graph the training and test misclassification errors vs tree depth (or log2 of nodes) as two separate curves. Report in a table the minimum test error and the tree depth (number of nodes or splits) for which the minimum was attained. (2 points)

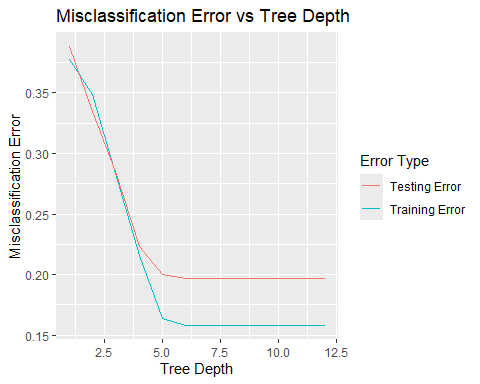
MADResults <- tibble(depth = integer(), train\_error = double(), test\_error = double())  
  
for (depth in 1:12) {  
 tree\_spec <- decision\_tree(tree\_depth = depth) %>%   
 set\_engine("rpart") %>%   
 set\_mode("classification")  
   
 tree\_fit <- tree\_spec %>%   
 fit(class ~ ., data = MADTrain)  
   
 train\_preds <- predict(tree\_fit, MADTrain, type = "class")  
 test\_preds <- predict(tree\_fit, MADValid, type = "class")  
   
 train\_error <- mean(train\_preds$.pred\_class != MADTrain$class)  
 test\_error <- mean(test\_preds$.pred\_class != MADValid$class)  
   
 MADResults <- MADResults %>%   
 add\_row(depth = depth, train\_error = train\_error, test\_error = test\_error)  
}

print(MADResults)

## # A tibble: 12 × 3  
## depth train\_error test\_error  
## <int> <dbl> <dbl>  
## 1 1 0.378 0.388  
## 2 2 0.349 0.335  
## 3 3 0.284 0.285  
## 4 4 0.216 0.223  
## 5 5 0.164 0.2   
## 6 6 0.158 0.197  
## 7 7 0.158 0.197  
## 8 8 0.158 0.197  
## 9 9 0.158 0.197  
## 10 10 0.158 0.197  
## 11 11 0.158 0.197  
## 12 12 0.158 0.197

* The minimum depth and validation error are given by 6 and 0.1966667 respectively.

ggplot(MADResults, aes(x = depth)) +  
 geom\_line(aes(y = train\_error, color = "Training Error")) +  
 geom\_line(aes(y = test\_error, color = "Testing Error")) +  
 labs(title = "Misclassification Error vs Tree Depth",  
 x = "Tree Depth",  
 y = "Misclassification Error",  
 color = "Error Type")



**b)** Repeat point a) on the satimage dataset. (2 points)

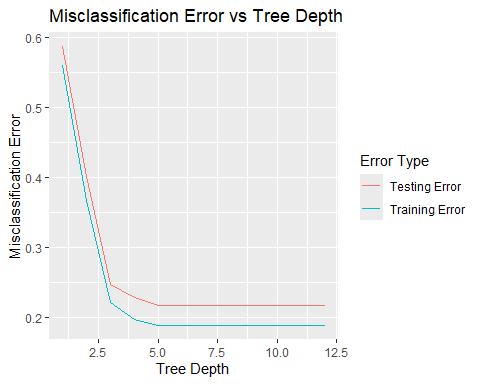
SATResults <- tibble(depth = integer(), train\_error = double(), test\_error = double())  
  
for (depth in 1:12) {  
 tree\_spec <- decision\_tree(tree\_depth = depth) %>%   
 set\_engine("rpart") %>%   
 set\_mode("classification")  
   
 tree\_fit <- tree\_spec %>%   
 fit(class ~ ., data = SATTrain)  
   
 train\_preds <- predict(tree\_fit, SATTrain, type = "class")  
 test\_preds <- predict(tree\_fit, SATTest, type = "class")  
   
 train\_error <- mean(train\_preds$.pred\_class != SATTrain$class)  
 test\_error <- mean(test\_preds$.pred\_class != SATTest$class)  
   
 SATResults <- SATResults %>%   
 add\_row(depth = depth, train\_error = train\_error, test\_error = test\_error)  
}

print(SATResults)

## # A tibble: 12 × 3  
## depth train\_error test\_error  
## <int> <dbl> <dbl>  
## 1 1 0.560 0.588  
## 2 2 0.368 0.401  
## 3 3 0.222 0.248  
## 4 4 0.197 0.229  
## 5 5 0.188 0.217  
## 6 6 0.188 0.217  
## 7 7 0.188 0.217  
## 8 8 0.188 0.217  
## 9 9 0.188 0.217  
## 10 10 0.188 0.217  
## 11 11 0.188 0.217  
## 12 12 0.188 0.217

* The minimum depth and testing error are given by 5 and 0.2170 respectively.

ggplot(SATResults, aes(x = depth)) +  
 geom\_line(aes(y = train\_error, color = "Training Error")) +  
 geom\_line(aes(y = test\_error, color = "Testing Error")) +  
 labs(title = "Misclassification Error vs Tree Depth",  
 x = "Tree Depth",  
 y = "Misclassification Error",  
 color = "Error Type")



**c)** On the madelon dataset, for each of k ∈ {3, 10, 30, 100, 300} train a random forest with k trees where the split attribute at each node is chosen from a random subset of 22 ≈ √500 features. Use the trained trees to predict the class labels on the training and test sets, and obtain the training and test misclassification errors. Plot on the same graph the training and test errors vs number of trees k as two separate curves. Report the training and test misclassification errors in a table. (2 points)

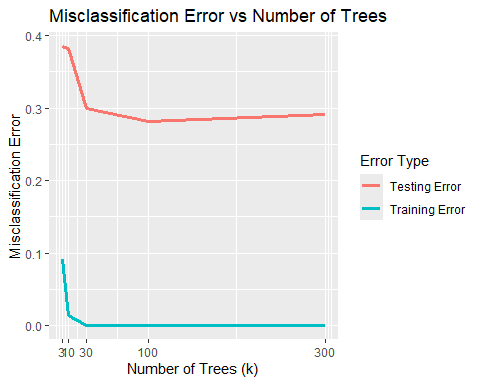
MADResults2 <- tibble(k = integer(), train\_error = double(), test\_error = double())  
  
for (k in c(3, 10, 30, 100, 300)) {  
 rf\_spec <- rand\_forest(trees = k, mtry = 22) %>%   
 set\_engine("ranger") %>%   
 set\_mode("classification")  
   
 rf\_fit <- rf\_spec %>%   
 fit(class ~ ., data = MADTrain)  
   
 train\_preds <- predict(rf\_fit, MADTrain, type = "class")  
 test\_preds <- predict(rf\_fit, MADValid, type = "class")  
   
 train\_error <- mean(train\_preds$.pred\_class != MADTrain$class)  
 test\_error <- mean(test\_preds$.pred\_class != MADValid$class)  
   
 MADResults2 <- MADResults2 %>%   
 add\_row(k = k, train\_error = train\_error, test\_error = test\_error)  
}

print(MADResults2)

## # A tibble: 5 × 3  
## k train\_error test\_error  
## <dbl> <dbl> <dbl>  
## 1 3 0.0905 0.385  
## 2 10 0.0145 0.382  
## 3 30 0 0.3   
## 4 100 0 0.282  
## 5 300 0 0.292

ggplot(MADResults2, aes(x = k)) +  
 geom\_line(aes(y = train\_error, color = "Training Error"), size = 1.2) +  
 geom\_line(aes(y = test\_error, color = "Testing Error"), size = 1.2) +  
 scale\_x\_continuous(breaks = c(3, 10, 30, 100, 300)) +  
 labs(title = "Misclassification Error vs Number of Trees",  
 x = "Number of Trees (k)",  
 y = "Misclassification Error",  
 color = "Error Type")

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



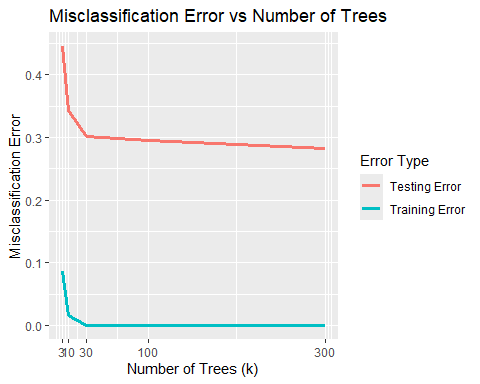
**d)** Repeat point c) on the madelon dataset where the split attribute at each node is chosen from a random subset of 9 ≈ log2(500) features. (1 point)

MADResults3 <- tibble(k = integer(), train\_error = double(), test\_error = double())  
  
for (k in c(3, 10, 30, 100, 300)) {  
 rf\_spec <- rand\_forest(trees = k, mtry = 22) %>%   
 set\_engine("ranger") %>%   
 set\_mode("classification")  
   
 rf\_fit <- rf\_spec %>%   
 fit(class ~ ., data = MADTrain)  
   
 train\_preds <- predict(rf\_fit, MADTrain, type = "class")  
 test\_preds <- predict(rf\_fit, MADValid, type = "class")  
   
 train\_error <- mean(train\_preds$.pred\_class != MADTrain$class)  
 test\_error <- mean(test\_preds$.pred\_class != MADValid$class)  
   
 MADResults3 <- MADResults3 %>%   
 add\_row(k = k, train\_error = train\_error, test\_error = test\_error)  
}

print(MADResults3)

## # A tibble: 5 × 3  
## k train\_error test\_error  
## <dbl> <dbl> <dbl>  
## 1 3 0.0855 0.445  
## 2 10 0.016 0.343  
## 3 30 0 0.302  
## 4 100 0 0.295  
## 5 300 0 0.282

ggplot(MADResults3, aes(x = k)) +  
 geom\_line(aes(y = train\_error, color = "Training Error"), size = 1.2) +  
 geom\_line(aes(y = test\_error, color = "Testing Error"), size = 1.2) +  
 scale\_x\_continuous(breaks = c(3, 10, 30, 100, 300)) +  
 labs(title = "Misclassification Error vs Number of Trees",  
 x = "Number of Trees (k)",  
 y = "Misclassification Error",  
 color = "Error Type")



**e)** Repeat point c) on the madelon dataset where the split attribute at each node is chosen from all 500 features. (1 point)

MADResults4 <- tibble(k = integer(), train\_error = double(), test\_error = double())  
  
for (k in c(3, 10, 30, 100, 300)) {  
 rf\_spec <- rand\_forest(trees = k, mtry = 22) %>%   
 set\_engine("ranger") %>%   
 set\_mode("classification")  
   
 rf\_fit <- rf\_spec %>%   
 fit(class ~ ., data = MADTrain)  
   
 train\_preds <- predict(rf\_fit, MADTrain, type = "class")  
 test\_preds <- predict(rf\_fit, MADValid, type = "class")  
   
 train\_error <- mean(train\_preds$.pred\_class != MADTrain$class)  
 test\_error <- mean(test\_preds$.pred\_class != MADValid$class)  
   
 MADResults4 <- MADResults4 %>%   
 add\_row(k = k, train\_error = train\_error, test\_error = test\_error)  
}

print(MADResults4)

## # A tibble: 5 × 3  
## k train\_error test\_error  
## <dbl> <dbl> <dbl>  
## 1 3 0.0975 0.45   
## 2 10 0.016 0.395  
## 3 30 0 0.307  
## 4 100 0 0.303  
## 5 300 0 0.258

ggplot(MADResults4, aes(x = k)) +  
 geom\_line(aes(y = train\_error, color = "Training Error"), size = 1.2) +  
 geom\_line(aes(y = test\_error, color = "Testing Error"), size = 1.2) +  
 scale\_x\_continuous(breaks = c(3, 10, 30, 100, 300)) +  
 labs(title = "Misclassification Error vs Number of Trees",  
 x = "Number of Trees (k)",  
 y = "Misclassification Error",  
 color = "Error Type")

