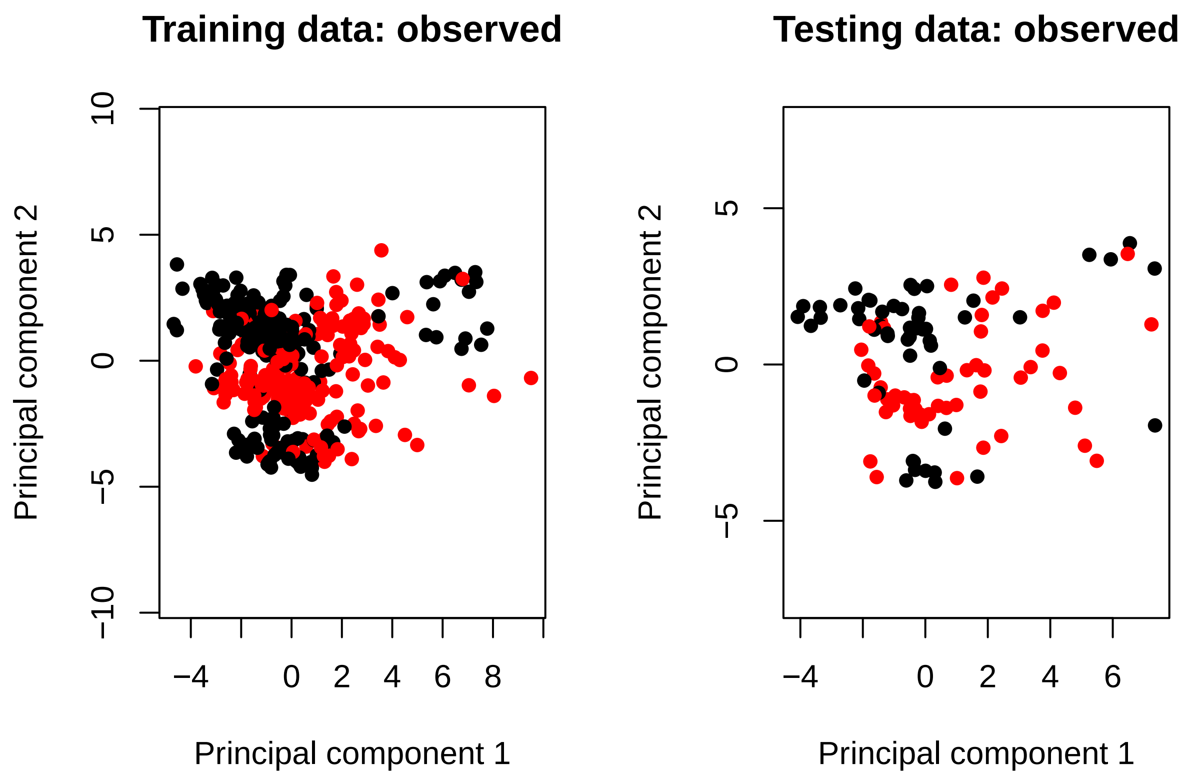
36-669 HW4

Question 1

```r
library(tidyverse)
library(glmnet)
library(xgboost)

snp <- read.csv("https://raw.githubusercontent.com/xuranw/469_public/master/hw4/synthetic_famuss.csv")
snp_data <- as.matrix(snp)
heart_disease <- snp_data[,1]
snp_data <- snp_data[,-1]

set.seed(10)
n <- length(heart_disease)
idx <- sample(1:n, round(.2*n))
train_dat <- snp_data[-idx,]
train_label <- heart_disease[-idx]
test_dat <- snp_data[idx,]
test_label <- heart_disease[idx]
```

1.A

```r
# scale. = T
pca_res <- stats::prcomp(snp_data, center = T, scale. = T)
pca1_train <- pca_res$x[-idx, 1]
pca2_train <- pca_res$x[-idx, 2]

pca1_test <- pca_res$x[idx, 1]
pca2_test <- pca_res$x[idx, 2]

par(mfrow = c(1,2))
plot(pca1_train, pca2_train, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(heart_disease[-idx] == 1, "red", "black"), main = "Training data: observed"

plot(pca1_test, pca2_test, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(heart_disease[idx] == 1, "red", "black"), main = "Testing data: observed")
```



```r
par(mfrow = c(1,1))
```

While there is some clustering for the black data points (i.e. no heart disease), there does not appear to be a pattern for the red data points clustering together (i.e. heart disease). There is no apparent underlying relationship in either the test or training data between the SNPs and having heart disease from plotting the data using the 2 largest principal components.

1.B

```r
set.seed(10)
cv_glm_train <- glmnet::cv.glmnet(x = train_dat, y = train_label, family = "binomial",
                                  alpha = 1, nfolds = 10, intercept = T)

# predict training error using training data
cv_pred_train <- as.numeric(glmnet:::predict.cv.glmnet(cv_glm_train, newx =  train_dat,
                                                        s = "lambda.1se", type = "class"))

# misclassification rate
tab_mis_train <- table(cv_pred_train, train_label)
tab_mis_train
```

```
##                train_label
## cv_pred_train   0   1
##              0 178 108
##              1  60  79
```

```
# # calculating the rate
1 - sum(diag(tab_mis_train))/sum(tab_mis_train)
```

```
## [1] 0.3952941
```

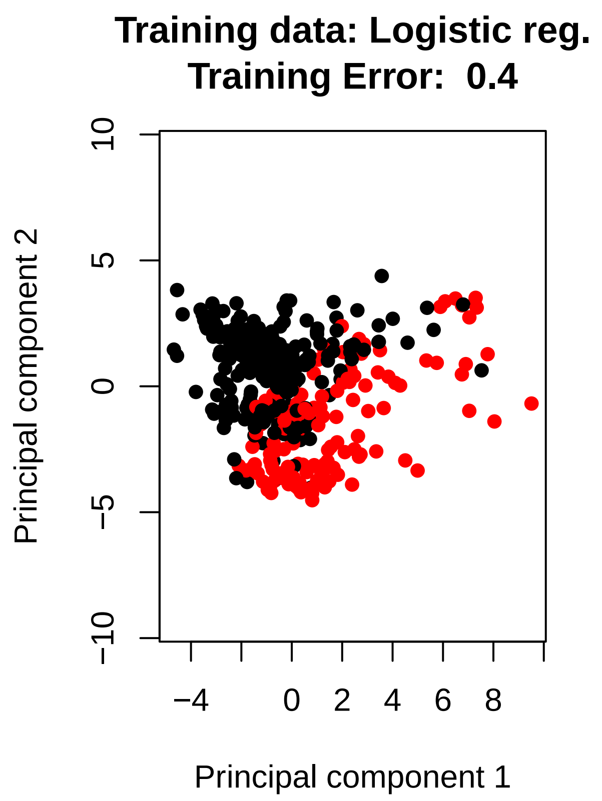
```
# predict training error using testing data
cv_pred_test <- as.numeric(glmnet:::predict.cv.glmnet(cv_glm_train, newx =  test_dat,
                                                     s = "lambda.1se", type = "class"))

tab_mis_test <- table(cv_pred_test, test_label)
tab_mis_test
```

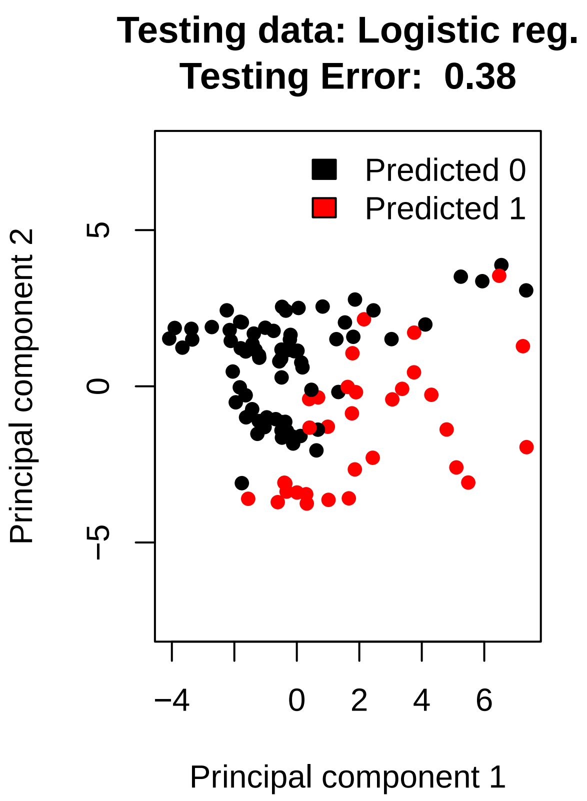
```
##              test_label
## cv_pred_test  0  1
##             0 43 31
##             1  9 23
```

```
# # calculating the rate
1 - sum(diag(tab_mis_test))/sum(tab_mis_test)
```

```
## [1] 0.3773585
```

```
par(mfrow = c(1,2))
plot(pca1_train, pca2_train, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(cv_pred_train == 1, "red", "black"),
     main = paste("Training data: Logistic reg.\nTraining Error: ", round(1 - sum(diag(tab_mis_train))/s
     ylim = c(-6, 6))

# fix the ylim labels
plot(pca1_test, pca2_test, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(cv_pred_test == 1, "red", "black"),
     main = paste("Testing data: Logistic reg.\nTesting Error: ", round(1 - sum(diag(tab_mis_test))/sum
     ylim = c(-6, 6))
legend("topright", legend = c("Predicted 0", "Predicted 1"), fill = c("black", "red"), bty = "n")
```





```
par(mfrow = c(1,1))
```

We see from both the training and test data that the groups are classified into 0 and 1 based on the logistic regression results by a roughly straight line boundary (a generalized linear model). Since the training and testing data are clustered together rather than separated by a straight line threshold, there is a large prediction error for both the training and testing data.

1.C

```
# XGBoost
set.seed(10)
xgb_cv <- xgboost::xgb.cv(data = train_dat, label = train_label, nrounds = 20, max.depth = 5, nfold = 5
```

```
## [1]  train-error:0.082343+0.006314   test-error:0.157471+0.035881
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 5 rounds.
##
## [2]  train-error:0.072924+0.008456   test-error:0.150716+0.024760
## [3]  train-error:0.068833+0.007663   test-error:0.145899+0.027684
## [4]  train-error:0.063530+0.004776   test-error:0.134161+0.024489
## [5]  train-error:0.059416+0.007084   test-error:0.129510+0.028441
## [6]  train-error:0.052341+0.008527   test-error:0.131836+0.027731
## [7]  train-error:0.049405+0.004986   test-error:0.138813+0.032728
## [8]  train-error:0.038817+0.005649   test-error:0.134161+0.030547
## [9]  train-error:0.030578+0.004711   test-error:0.138784+0.031796
## [10] train-error:0.025289+0.007554   test-error:0.138757+0.026883
## Stopping. Best iteration:
## [5]  train-error:0.059416+0.007084   test-error:0.129510+0.028441
```

```
# best iteration from xgboost cv
xgb_cv$best_iteration
```

```
## [1] 5
```

```
# using the best iteration from cross validation for the model
xgb_fit <- xgboost::xgboost(data = train_dat, label = train_label, nrounds = xgb_cv$best_iteration, max
```

```
## [16:28:14] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:14] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.515663
## [2]  train-logloss:0.417162
## [3]  train-logloss:0.344497
## [4]  train-logloss:0.297460
## [5]  train-logloss:0.257092
```

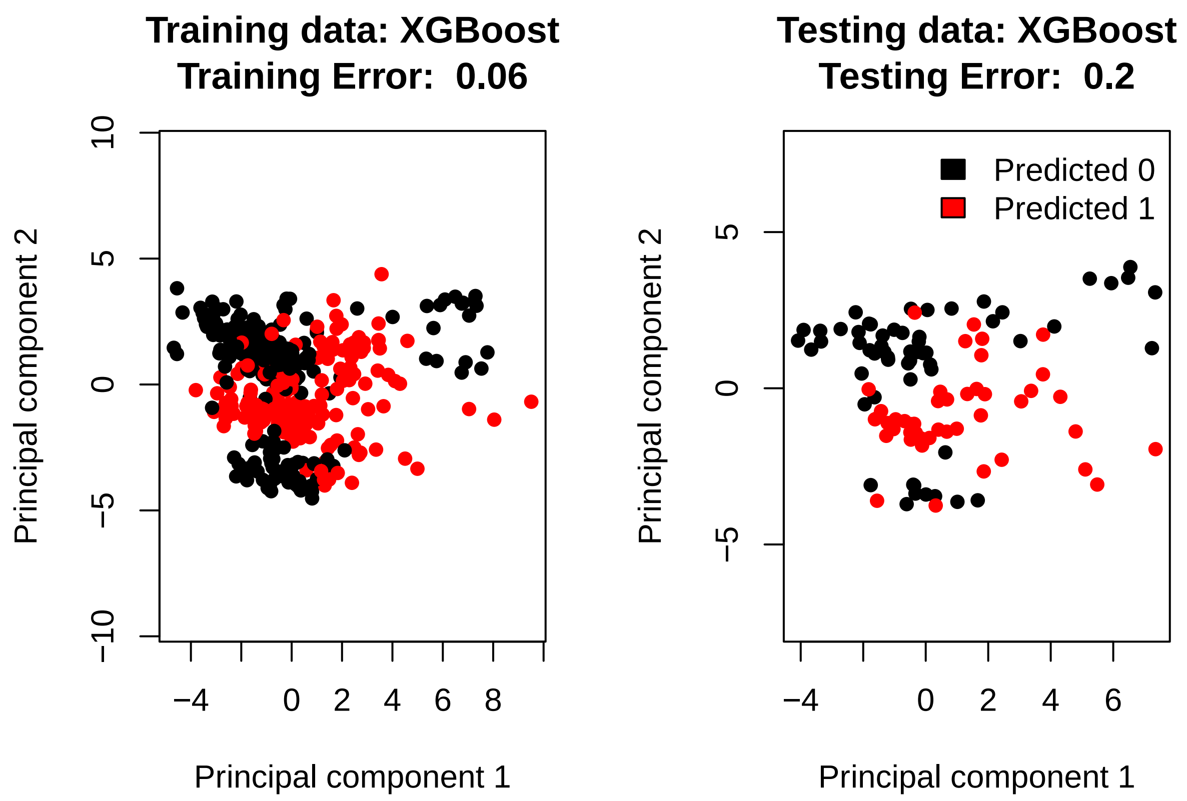
```
xgb_train_pred <- as.numeric(stats::predict(xgb_fit, newdata = train_dat) > 0.5)
tab_xgb_train <- table(train_label, xgb_train_pred)
1-max(c(sum(diag(tab_xgb_train)), tab_xgb_train[1,2]+tab_xgb_train[2,1]))/sum(tab_xgb_train)
```

```
## [1] 0.06117647
```

```
xgb_test_pred <- as.numeric(stats::predict(xgb_fit, newdata = test_dat) > 0.5)
tab_xgb_test <- table(test_label, xgb_test_pred)
1-max(c(sum(diag(tab_xgb_test)), tab_xgb_test[1,2]+tab_xgb_test[2,1]))/sum(tab_xgb_test)
```

```
## [1] 0.1981132
```

```
par(mfrow = c(1,2))
plot(pca1_train, pca2_train, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(xgb_train_pred == 1, "red", "black"),
     main = paste("Training data: XGBoost\nTraining Error: ",
                  round(1-max(c(sum(diag(tab_xgb_train)), tab_xgb_train[1,2]+tab_xgb_train[2,1]))/sum(ta

# fix the labels
plot(pca1_test, pca2_test, asp = T, xlab = "Principal component 1", ylab = "Principal component 2",
     pch = 16, col = ifelse(xgb_test_pred == 1, "red", "black"),
     main = paste("Testing data: XGBoost\nTesting Error: ",
                  round(1-max(c(sum(diag(tab_xgb_test)), tab_xgb_test[1,2]+tab_xgb_test[2,1]))/sum(tab_
legend("topright", legend = c("Predicted 0", "Predicted 1"), fill = c("black", "red"), bty = "n")
```



```
par(mfrow = c(1,1))
```

Comparing Figures 2 and 3, we see that both the training and testing error for the XGBoost method is lower compared to Logistic Regression since XGBoost more appropriately predicts the data since there is not a straight-line boundary between the people with and without heart disease. This is visually assessed by comparing the similarities in the colors for Figures 2 and 3 when looking at Figure 1. The colors are much more closely aligned between Figures 1 and 3 compared to Figures 1 and 2. This illustrates that XGBoost better predicts the data than Logistic Regression since it can learn to predict data with non-linear boundaries, unlike Logistic Regression which only can make linear boundaries.

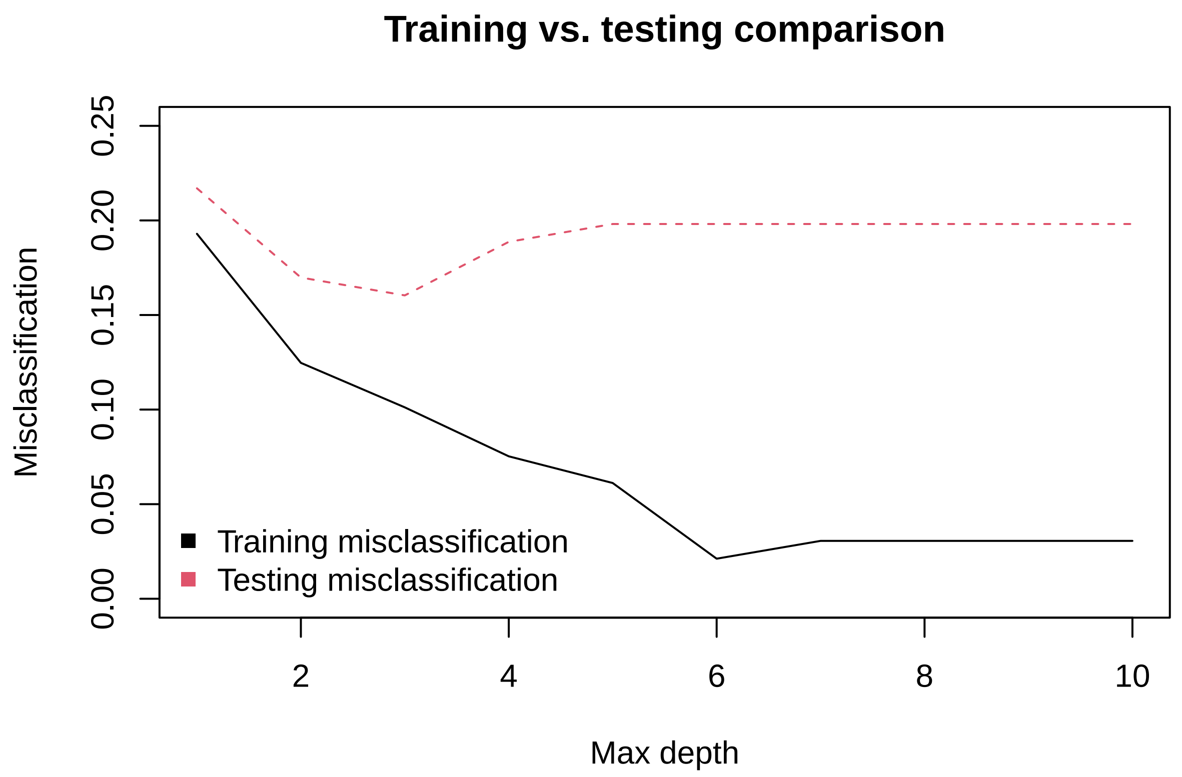
1.D

```
loop_cv <- NULL
loop_fit <- NULL
train_loop <- rep(0, 10)
test_loop <- rep(0, 10)

set.seed(10)
for (i in 1:10) {
        set.seed(10)
        loop_cv <- xgboost::xgb.cv(data = train_dat, label = train_label, nrounds = 20,
                                   nfold = 5, metrics = list("error"), max.depth = i,
                                   objective = "binary:logistic", early_stopping_rounds = 5,
                                verbose = F)
        loop_fit <- xgboost::xgboost(data = train_dat, label = train_label,
                                  nrounds = loop_cv$best_iteration, nfold = 5,
                                  max.depth = i, objective = "binary:logistic",
                                  verbose = F, eval_metric = "logloss")
        # predicting temporary variable
        temp_pred <- as.numeric(stats::predict(loop_fit, newdata = train_dat) > 0.5)
        tab_temp <- table(train_label, temp_pred)
        train_loop[i] <- 1-max(c(sum(diag(tab_temp)), tab_temp[1,2]+tab_temp[2,1]))/sum(tab_temp)

        temp_pred <- as.numeric(stats::predict(loop_fit, newdata = test_dat) > 0.5)
        tab_temp <- table(test_label, temp_pred)
        test_loop[i] <- 1-max(c(sum(diag(tab_temp)), tab_temp[1,2]+tab_temp[2,1]))/sum(tab_temp)
}
```

```
## [16:28:14] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:15] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:15] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:16] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:16] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
```

```
##
##
## [16:28:16] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:17] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:18] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:18] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:19] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
```

```r
plot(train_loop, type = 'l', ylim = c(0, 0.25), xlab = 'Max depth',
     ylab = 'Misclassification', main = 'Training vs. testing comparison')
lines(test_loop, lty = 2, col = 2)
legend('bottomleft',c('Training misclassification', 'Testing misclassification'),
       pch = c(15,15), col = c(1,2), bty='n')
```



We see that as the maximum depth of a tree increases for the training data, the training and testing misclassification rates decrease up to a point. The errors then begin to increase as maximum depth for the trees increase, suggest that there are diminishing benefits for increasing the max depth. This is indicative of the overfitting phenomenon, as trying to overfit the model using the training data has noticeable effects on the misclassification of the testing data since the model fits the noise of the training data which is not present in the testing data.

Based on figure 4, we see that a maximum depth of 3 is a good choice because it is associated with the minimum misclassification rate for the testing data and is not overfitting the noise in the training data, which is more likely to fit the noise of the data.

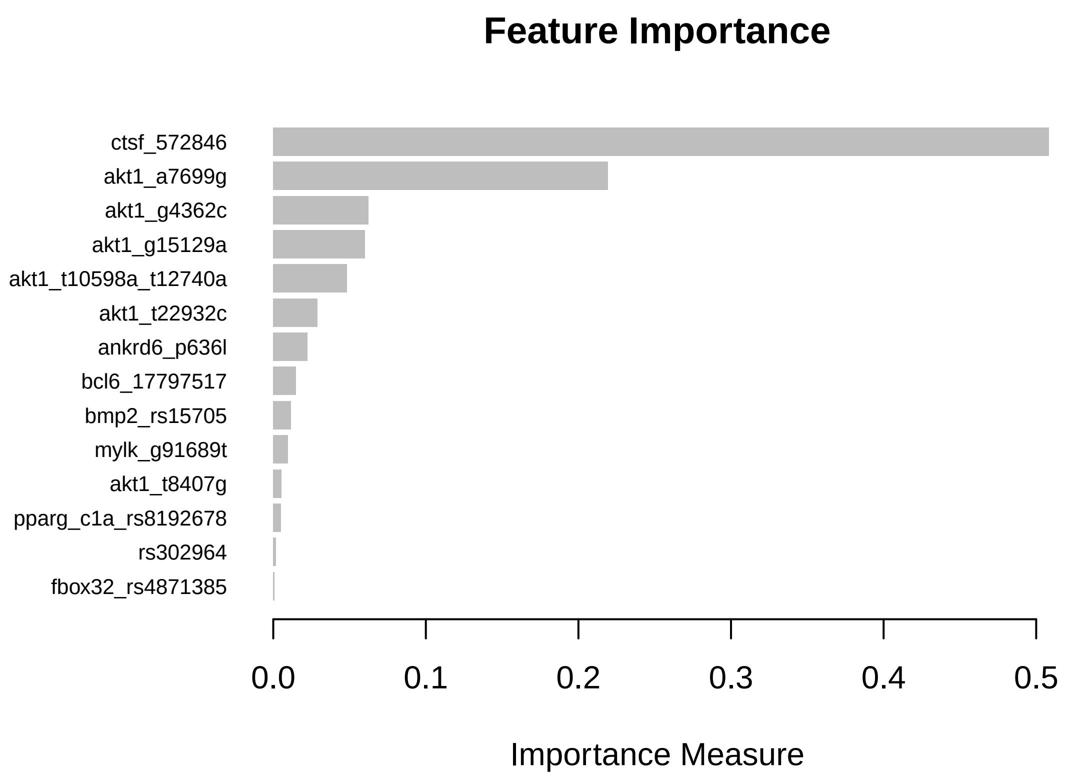
1.E

```
xgb_fit_3 <- xgboost::xgboost(data = train_dat, label = train_label,
                              nrounds = xgb_cv$best_iteration, nfold = 5,
                              max.depth = 3, objective = "binary:logistic")
```

```
## [16:28:19] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
```

```
##
## [16:28:19] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.549396
## [2]  train-logloss:0.466652
## [3]  train-logloss:0.414189
## [4]  train-logloss:0.378304
## [5]  train-logloss:0.356117
```

```
importance_mat <- xgboost::xgb.importance(model = xgb_fit_3)
xgboost::xgb.plot.importance(importance_mat, main = "Feature Importance", xlab = "Importance Measure")
```



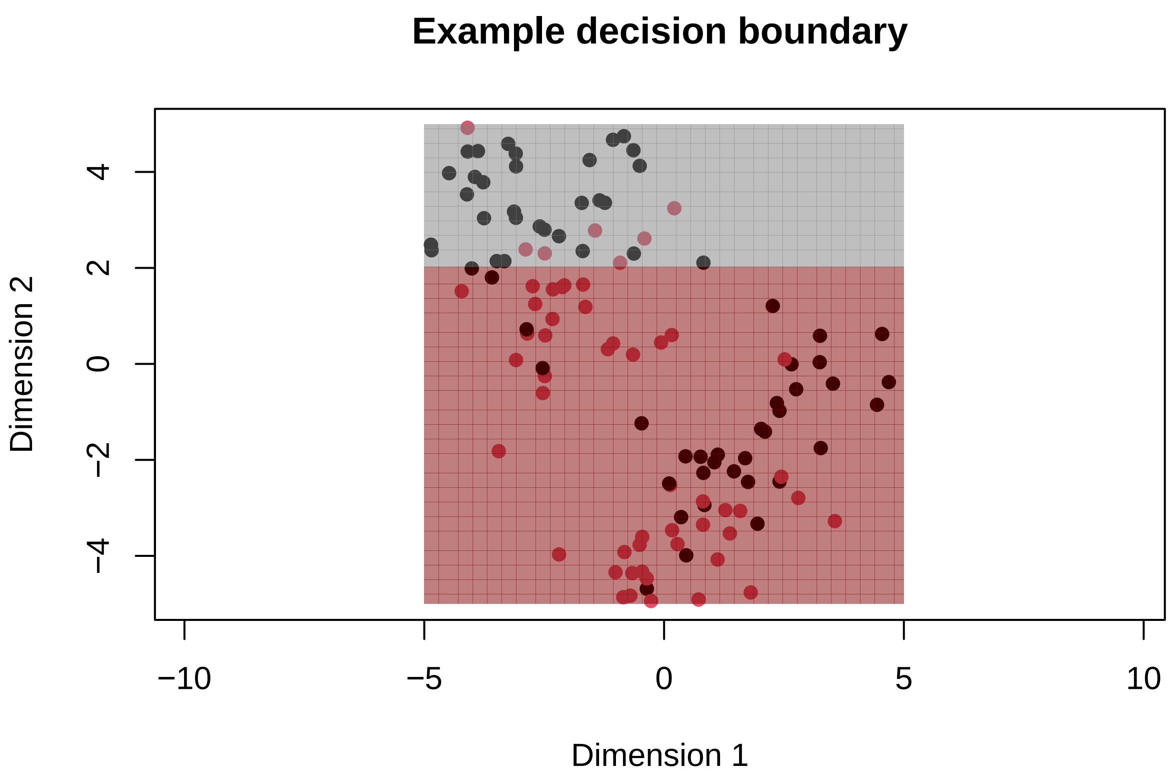
```
head(importance_mat)
```

```
##                  Feature       Gain      Cover  Frequency Importance
## 1:           ctsf_572846 0.50830395 0.47988975 0.25714286 0.50830395
## 2:           akt1_a7699g 0.21946432 0.18022926 0.17142857 0.21946432
## 3:           akt1_g4362c 0.06254541 0.03634012 0.08571429 0.06254541
## 4:          akt1_g15129a 0.06020718 0.03935084 0.08571429 0.06020718
## 5: akt1_t10598a_t12740a 0.04831598 0.03578289 0.08571429 0.04831598
## 6:          akt1_t22932c 0.02887719 0.01498356 0.02857143 0.02887719
```

"Gain" represents the fractional contribution of each feature to the model based on the total gain of this feature's splits; a higher percentage means a more important predictive feature. "Cover" is a metric of the number of observations related to this feature. "Frequency" is the percentage representing of the relative number of times a feature have been used in trees.

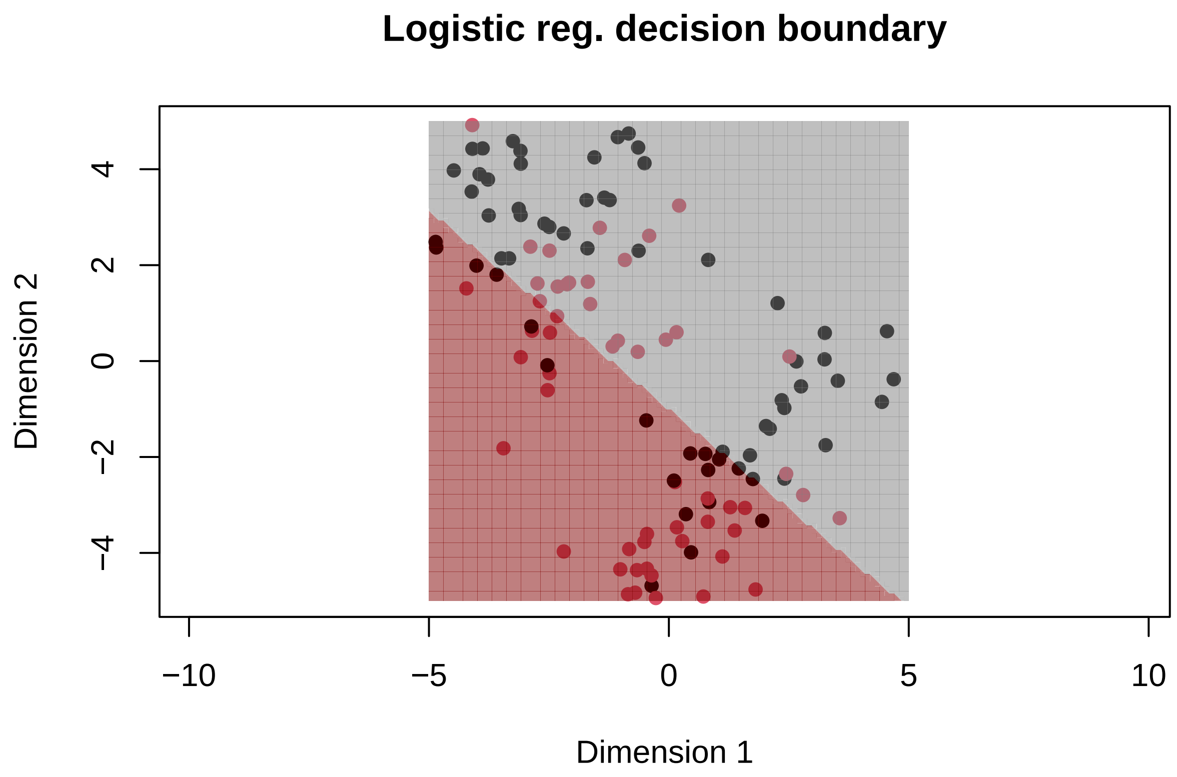
Question 2

```
source("https://raw.githubusercontent.com/xuranw/469_public/master/hw4/hw4_functions.R")
dat <- as.matrix(read.csv("https://raw.githubusercontent.com/xuranw/469_public/master/hw4/synthetic_data
y <- dat[,1]
x <- dat[,2:3]

grid_val <- seq(-5, 5, length.out = 100)
test_grid <- as.matrix(expand.grid(grid_val, grid_val))
colnames(test_grid) <- c("x1", "x2")

# using the practice code to see how predict plot works
example_classifier <- function(vec){
ifelse(vec[2] >= 2, 0, 1)
}
pred_vec <- apply(test_grid, 1, example_classifier)
plot_prediction_region(x, y, pred_vec, test_grid, xlab = "Dimension 1", ylab = "Dimension 2",
                       main = "Example decision boundary", pch = 16, asp = T)
```



2.A

```
# used glmnet to get prediction to work correctly (issue with vector length when predicting stats::glm)
set.seed(10)
glm_res <- stats::glm(y~x1+x2, data = data.frame(dat), family = stats::binomial)

pred_vec <- as.numeric(stats::predict(glm_res, type = "response", newdata = data.frame(test_grid)) > 0.

plot_prediction_region(x = x, y = y, pred_vec = pred_vec, test_grid = test_grid,
                       xlab = "Dimension 1", ylab = "Dimension 2",
                       main = "Logistic reg. decision boundary", pch = 16, asp = T)
```



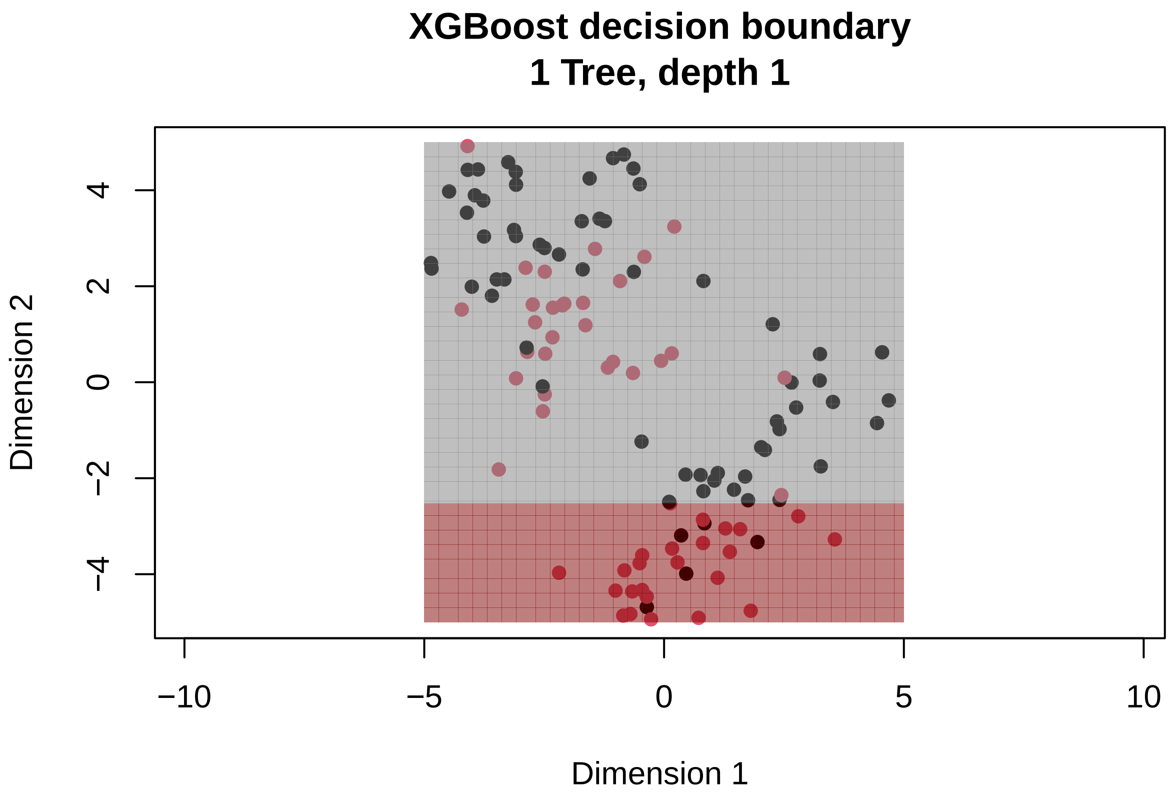
The figure shows that logistic regression crates a straight-line decision boundary to classify the points, and we see that this classification method is not very accurate compared to the actual data and true classification boundary. The appropriate decision boundary for the points is a non-linear boundary line, so many of the points are misclassified when using logistic regression.

2.B

```
xgb_fit2 <- xgboost::xgboost(data = x, label = y, max.depth = 1, nround = 1, objective = "binary:logisti
```

```
## [16:28:20] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.647286
```

```r
pred_vec2 <- as.numeric(predict(xgb_fit2, test_grid) > 0.5)

plot_prediction_region(x = x, y = y, pred_vec = pred_vec2, test_grid = test_grid,
                       xlab = "Dimension 1", ylab = "Dimension 2",
                       main = "XGBoost decision boundary\n1 Tree, depth 1",
                       pch = 16, asp = T)
```



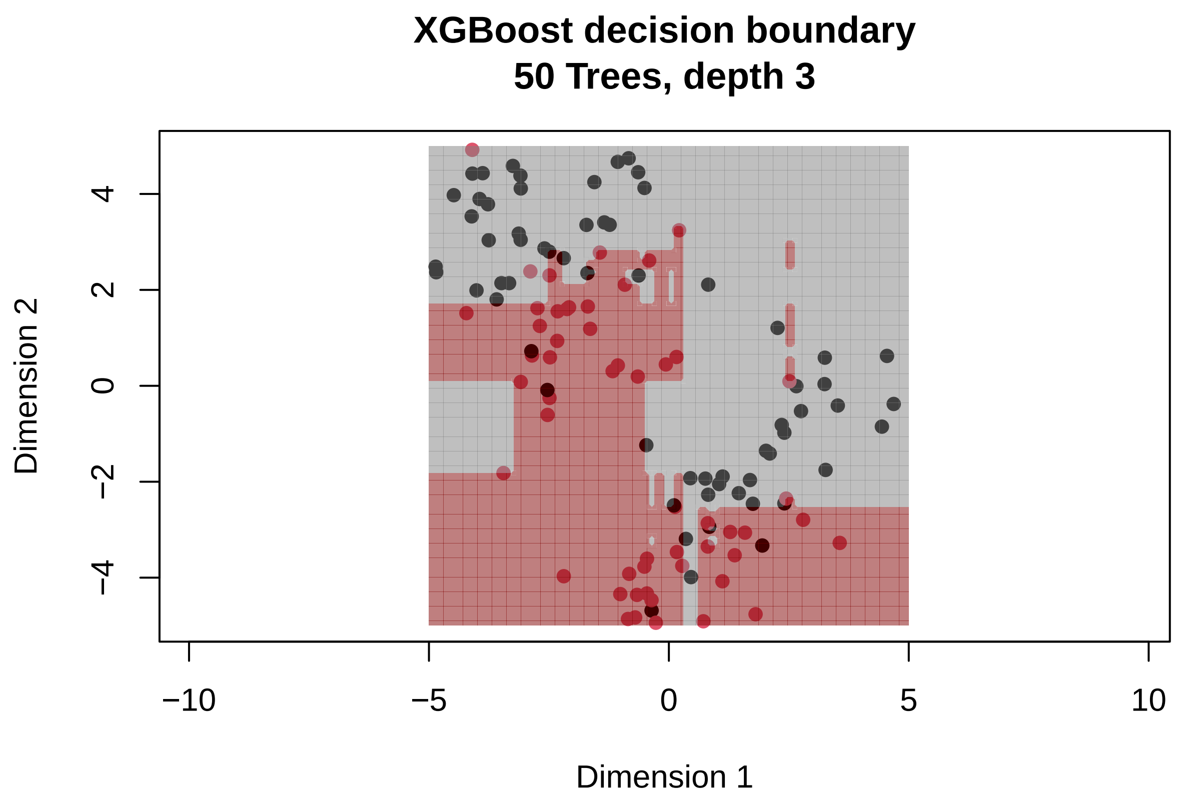
Using a single decision tree, we see that the decision boundary for classification is a horizontal line at a value slightly below -2, which minimizes the classification error for a single split. This is not very accurate when compared to the data and true classification boundary, and not as useful at predicting compared to logistic regression.

2.C

```r
xgb_fit3 <- xgboost::xgboost(data = x, label = y, max.depth = 3,
                             nround = 50, objective = "binary:logistic")
```

```
## [16:28:20] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.579950
## [2]  train-logloss:0.516581
## [3]  train-logloss:0.468614
## [4]  train-logloss:0.430963
## [5]  train-logloss:0.400332
## [6]  train-logloss:0.376947
## [7]  train-logloss:0.361639
## [8]  train-logloss:0.346201
## [9]  train-logloss:0.334063
## [10] train-logloss:0.323071
## [11] train-logloss:0.315519
## [12] train-logloss:0.311264
## [13] train-logloss:0.304253
## [14] train-logloss:0.299021
## [15] train-logloss:0.296161
## [16] train-logloss:0.290397
## [17] train-logloss:0.285946
## [18] train-logloss:0.281914
## [19] train-logloss:0.279954
## [20] train-logloss:0.276330
## [21] train-logloss:0.273982
## [22] train-logloss:0.269900
## [23] train-logloss:0.265266
## [24] train-logloss:0.262186
## [25] train-logloss:0.257441
## [26] train-logloss:0.254821
## [27] train-logloss:0.249201
## [28] train-logloss:0.245359
## [29] train-logloss:0.243088
## [30] train-logloss:0.240282
## [31] train-logloss:0.238308
## [32] train-logloss:0.235701
## [33] train-logloss:0.231800
## [34] train-logloss:0.228814
## [35] train-logloss:0.227408
## [36] train-logloss:0.225558
## [37] train-logloss:0.220618
## [38] train-logloss:0.217914
## [39] train-logloss:0.216267
## [40] train-logloss:0.215144
## [41] train-logloss:0.213224
## [42] train-logloss:0.209986
## [43] train-logloss:0.208676
## [44] train-logloss:0.206995
## [45] train-logloss:0.205786
## [46] train-logloss:0.204323
## [47] train-logloss:0.201676
## [48] train-logloss:0.199520
## [49] train-logloss:0.197981
## [50] train-logloss:0.195705
```

```r
pred_vec3 <- as.numeric(predict(xgb_fit3, test_grid) > 0.5)

plot_prediction_region(x = x, y = y, pred_vec = pred_vec3, test_grid = test_grid,
                       xlab = "Dimension 1", ylab = "Dimension 2",
                       main = "XGBoost decision boundary\n50 Trees, depth 3",
                       pch = 16, asp = T)
```



We see that boundary for the complicated tree is broken up into many sections as it tries to classify the noise of the data. While the classification is more accurate than the single tree or logistic regression classification approaches, it also is excessively complex since it has small sections of classification that doesn't only include the signal but also the noise. This suggests that the model is overfitting the training data, and will likely have a large classification error if tested on other data.

2.D

```
set.seed(10)
xgb_cv2 <- xgboost::xgb.cv(data = x, label = y,
                           nrounds = 20, nfold = 5, metrics = list("error"),
                           max_depth = 3, objective = "binary:logistic",
                           early_stopping_rounds = 5)
```

```
## [1]  train-error:0.155983+0.010826   test-error:0.231522+0.061106
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 5 rounds.
##
## [2]  train-error:0.147449+0.010582   test-error:0.205435+0.019699
## [3]  train-error:0.147449+0.010582   test-error:0.231522+0.071574
## [4]  train-error:0.136788+0.012804   test-error:0.188043+0.042600
## [5]  train-error:0.138916+0.007257   test-error:0.222826+0.075307
## [6]  train-error:0.123953+0.005682   test-error:0.222101+0.031033
## [7]  train-error:0.123953+0.008862   test-error:0.230435+0.041301
## [8]  train-error:0.117547+0.007178   test-error:0.213406+0.024316
## [9]  train-error:0.123976+0.011375   test-error:0.222101+0.031033
## Stopping. Best iteration:
## [4]  train-error:0.136788+0.012804   test-error:0.188043+0.042600
```

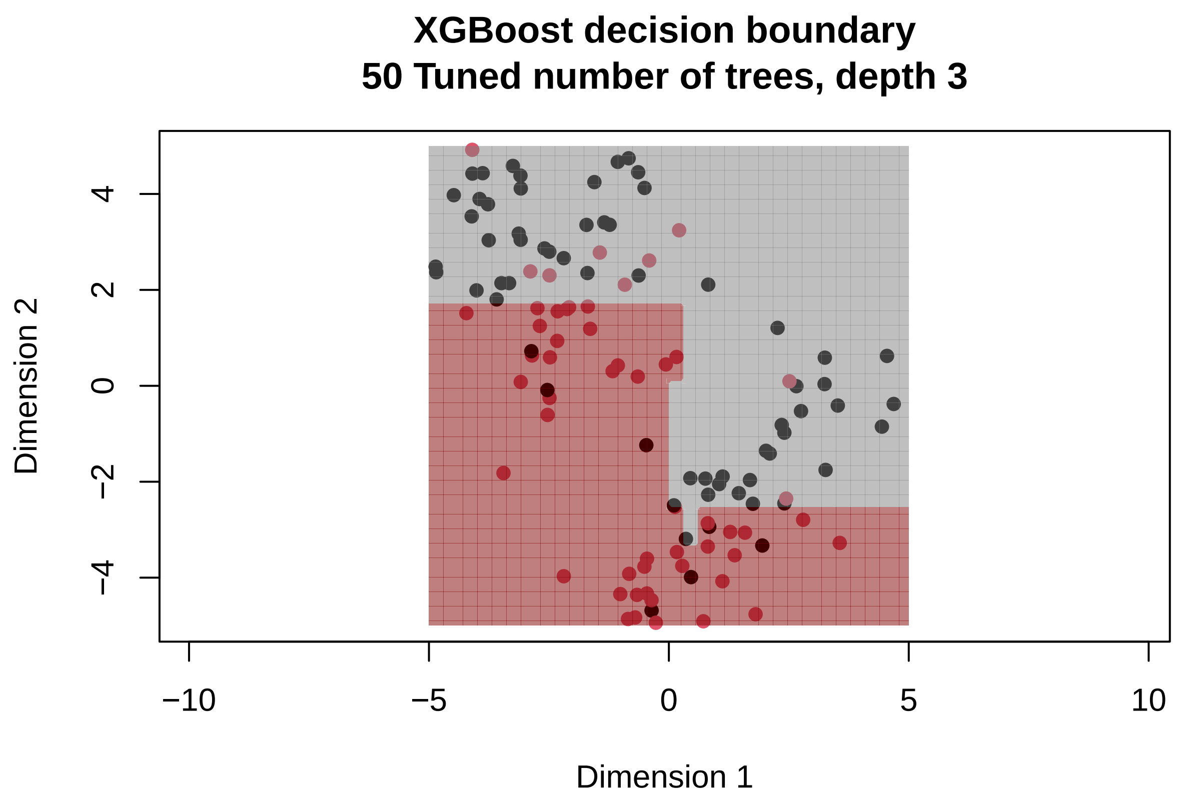
```
xgb_cv2$best_iteration
```

```
## [1] 4
```

```
xgb_fit4 <- xgboost::xgboost(data = x, label = y,
                            nrounds = xgb_cv2$best_iteration, nfold = 5,
                            max_depth = 3, objective = "binary:logistic")
```

```
## [16:28:20] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "nfold" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [16:28:20] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
## [1]  train-logloss:0.579950
## [2]  train-logloss:0.516581
## [3]  train-logloss:0.468614
## [4]  train-logloss:0.430963
```

```
pred_vec4 <- as.numeric(stats::predict(xgb_fit4, newdata = test_grid) > 0.5)


plot_prediction_region(x = x, y = y, pred_vec = pred_vec4, test_grid = test_grid,
                       xlab = "Dimension 1", ylab = "Dimension 2",
                       main = "XGBoost decision boundary\n50 Tuned number of trees, depth 3",
                       pch = 16, asp = T)
```



The boundary for the tuned tree provides a better fit for the data than the logistic regression classification or the simple tree classification. While it's fit is not as accurate compared to the complex tree, it has a more smooth and continuous area since it classifies the training data without overfitting the noise. This suggests the model does not suffer as much from overfitting compared to the more complex model. The method also most closely resembles the actual boundaries for the data, which implies that it is the most appropriate classification method for the data of the ones we've tried.