hw_2

Hengle Li

2/1/2020

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(broom)
library(rcfss)
library(rsample)
library(patchwork)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(ISLR)
library(yardstick)
```

```
## For binary classification, the first factor level is assumed to be the event.
## Set the global option `yardstick.event_first` to `FALSE` to change this.

##
## Attaching package: 'yardstick'

## The following object is masked from 'package:readr':
##
##     spec
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'

## The following objects are masked from 'package:yardstick':
##
##     precision, recall

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:patchwork':
##
##     area

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(class)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
set.seed(123)
options(digits = 3)
```

The Bayesian Classifier

Question 1

generate the sample

```r
#use runif given X1 and X2 are random uniform variables
sample200 <- tibble(X1 = runif(200, -1, 1),
                    X2 = runif(200, -1, 1),
#use rnorm to add error in the sample as normal distribution
                    e = rnorm(200, 0, 0.5))
#calculate Y by definition
sample200 <- sample200 %>%
  mutate(log_Y = X1 + X1^2 + X2 + X2^2 + e) %>%
  mutate(Y = logit2prob(log_Y)) %>%
#transform log-odds to probability
  mutate(result = ifelse(Y > 0.5, "success", "fail"))
```

create the naive Bayesian model

```r
#set up x and y
variables <- sample200 %>%
  dplyr::select(X1, X2)
x <- variables
y <- sample200$result
#train model
nb_model <- train(
  x = x,
  y = y,
  method = "nb"
)
```

```r
#predicted rate of success
nbpred1 <- predict(nb_model, newdata = sample200)
suc_count <- table(nbpred1)
sucs_rate <- suc_count[names(suc_count) == "success"] / 200
sucs_rate
```

```
## success
##     0.7
```

```r
#accuracy of the model
confusionMatrix(nb_model)
```

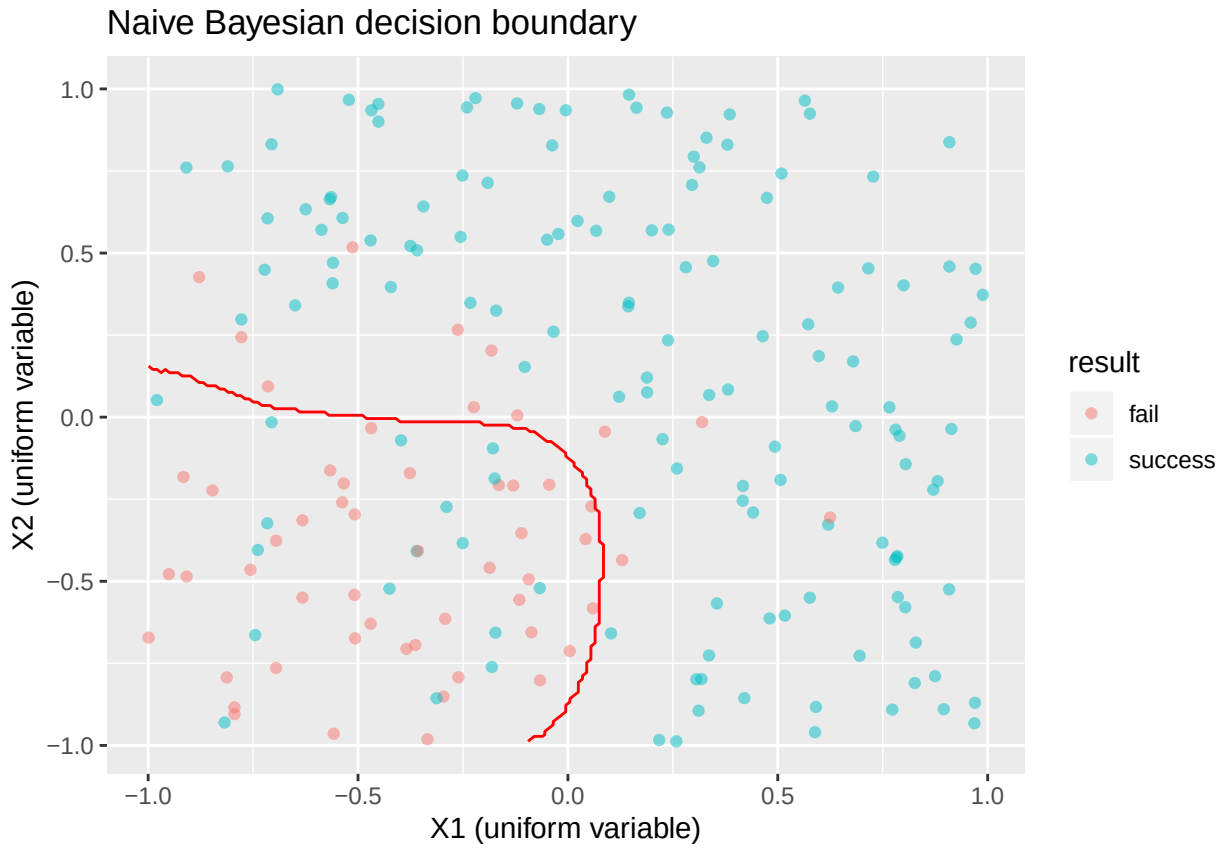
```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction fail success
##    fail    17.9     7.3
##    success 10.0    64.8
##
##  Accuracy (average) : 0.8272
```

create a new set of data for testing

```r
X1 <- seq(min(sample200$X1), max(sample200$X1), length = 200)
X2 <- seq(min(sample200$X2), max(sample200$X2), length = 200)
test_set <- expand.grid(X1 = X1, X2 = X2)
```

calculate a decision boundary with the test set and plot it against the training set

```r
#join the needed data first
pred_graph <- cbind(test_set, pred = predict(nb_model, test_set))

sample200 %>%
  ggplot(aes(x = X1, y = X2, color = result)) +
  geom_point(alpha = .5) +
  geom_contour(data = pred_graph,
               aes(z = as.numeric(pred)),
               color = "red",
               breaks = c(1.5)) +
  labs(title = "Naive Bayesian decision boundary",
       x = "X1 (uniform variable)",
       y = "X2 (uniform variable)")
```



Differences between LDA and QDA

question 2

First, train a single model to see how things work.

```
#stimulate data
sample1000 <- tibble(X1 = runif(1000, -1, 1),
                     X2 = runif(1000, -1, 1),
                     e = rnorm(1000, 0, 1))
sample1000_split <- initial_split(sample1000, prop = 0.7)
train_2 <- training(sample1000_split)
test_2 <- testing(sample1000_split)
#define success and failure
train_2 <- train_2 %>%
  mutate(Y = X1 + X2 + e) %>%
  mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
test_2 <- test_2 %>%
  mutate(Y = X1 + X2 + e) %>%
  mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
```

```
#train LDA model
X_q2 <- train_2 %>%
  dplyr::select(X1, X2)
Y_q2 <- train_2$result
lda2 <- train(
  x = X_q2,
  y = Y_q2,
  method = "lda"
)
#fitting in train data
confusionMatrix(lda2)
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  40.4 14.5
##      TRUE   12.8 32.3
##
##  Accuracy (average) : 0.7272
```

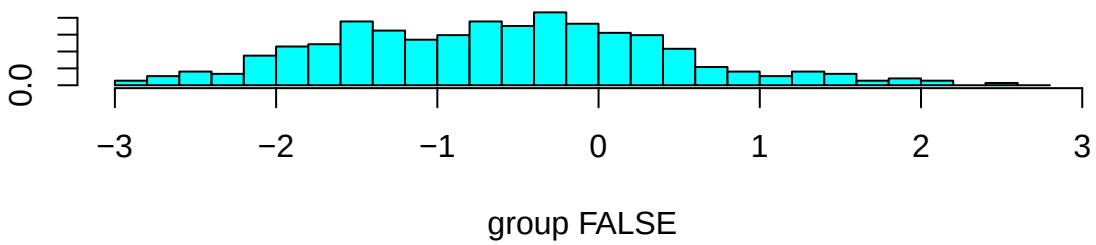
```
#fitting in test data
confusionMatrix(as.factor(test_2$result), predict(lda2,test_2))
```

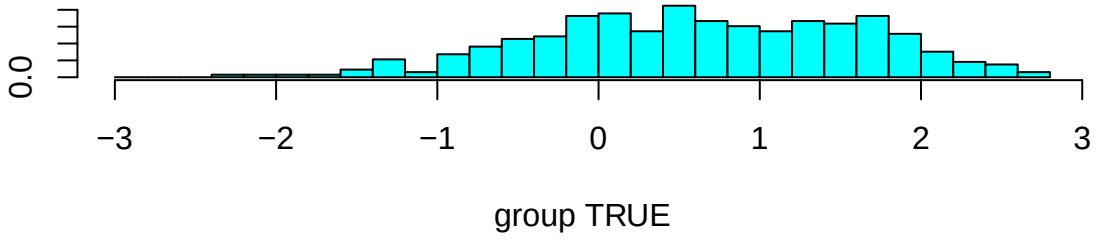
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE    97   33
```

```
##      TRUE     49  121
##
##                Accuracy : 0.727
##                  95% CI : (0.672, 0.776)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : 3.7e-14
##
##                   Kappa : 0.451
##
##  Mcnemar's Test P-Value : 0.0976
##
##             Sensitivity : 0.664
##             Specificity : 0.786
##          Pos Pred Value : 0.746
##          Neg Pred Value : 0.712
##              Prevalence : 0.487
##          Detection Rate : 0.323
##    Detection Prevalence : 0.433
##       Balanced Accuracy : 0.725
##
##        'Positive' Class : FALSE
##
```

Another way to construct the LDA model

```
lda22 <- lda(result ~ X1 + X2, data = train_2)
plot(lda22)
```





LDA predictions

```
ldapred2 <- predict(lda22, newdata = test_2)
#a look at the prediction result can show that lda2 and lda22 are the same model
#but they are not equally applicable to all functions, such as plot() and confusionMatrix()
table(data = test_2$result, predt = ldapred2$class)
```

```
##        predt
## data    FALSE TRUE
##   FALSE    97   33
##   TRUE     49  121
```

```
#error rate of the lda model
lda_error2 <- mean(test_2$result != ldapred2$class)
lda_error2
```

```
## [1] 0.273
```

Training the QDA model goes through the same procedures. Note that qda models do not work on `plot` function. Now proceed to creating the function for repeated sampling and testing.

```
#construct the function for mapping
repeat_test2 <- function(x){
  set.seed(x)
  sample1000 <- tibble(X1 = runif(1000, -1, 1),
```

```
                  X2 = runif(1000, -1, 1),
                  e = rnorm(1000, 0, 1))
  sample1000_split <- initial_split(sample1000, prop = 0.7)
  train_2 <- training(sample1000_split) %>%
    mutate(Y = X1 + X2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  test_2 <- testing(sample1000_split) %>%
    mutate(Y = X1 + X2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  lda22 <- lda(result ~ X1 + X2, data = train_2)
  qda22 <- qda(result ~ X1 + X2, data = train_2)
  ldapred_train2 <- predict(lda22, newdata = train_2)
  ldapred_test2 <- predict(lda22, newdata = test_2)
  lda_train_error2 <- mean(train_2$result != ldapred_train2$class)
  lda_test_error2 <- mean(test_2$result != ldapred_test2$class)
  qdapred_train2 <- predict(qda22, newdata = train_2)
  qdapred_test2 <- predict(qda22, newdata = test_2)
  qda_train_error2 <- mean(train_2$result != qdapred_train2$class)
  qda_test_error2 <- mean(test_2$result != qdapred_test2$class)
  results2 <- data.frame("test_count" = as.factor(x),
            "LDA_training_error"  = lda_train_error2,
            "LDA_testing_error" = lda_test_error2,
            "QDA_training_error"  = qda_train_error2,
            "QDA_testing_error" = qda_test_error2)
  results2
}
```

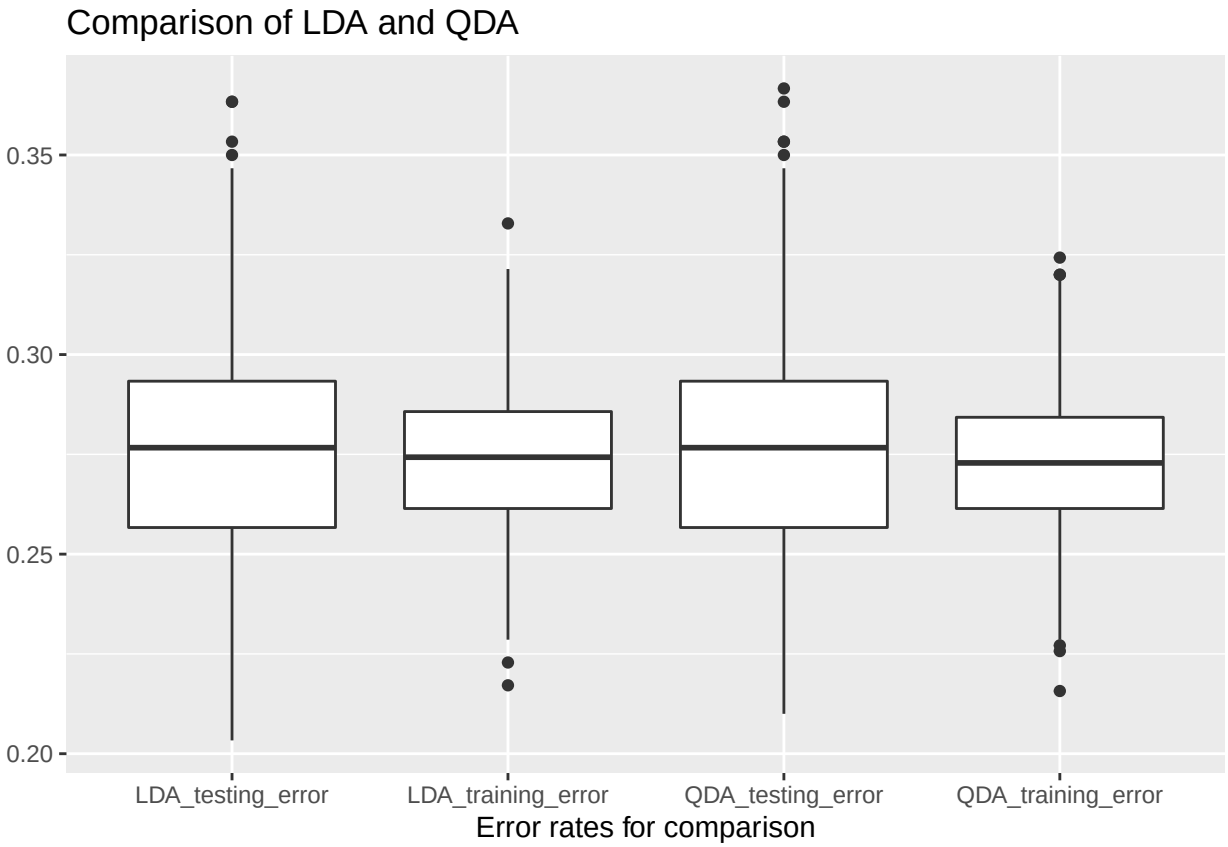
```
#mapping the results and joining them into a data frame
results1000_2 <- map(1:1000, repeat_test2) %>%
  bind_rows()
```

```
#summarize the results
results1000_2 %>%
  dplyr::select(-test_count) %>%
  summary() %>%
  as.table()
```

```
##  LDA_training_error LDA_testing_error QDA_training_error QDA_testing_error
##  Min.   :0.217      Min.   :0.203     Min.   :0.216      Min.   :0.210
##  1st Qu.:0.261      1st Qu.:0.257     1st Qu.:0.261      1st Qu.:0.257
##  Median :0.274      Median :0.277     Median :0.273      Median :0.277
##  Mean   :0.273      Mean   :0.276     Mean   :0.273      Mean   :0.277
##  3rd Qu.:0.286      3rd Qu.:0.293     3rd Qu.:0.284      3rd Qu.:0.293
##  Max.   :0.333      Max.   :0.363     Max.   :0.324      Max.   :0.367
```

```
#remove test_count before graphing the boxplot
results1000_2 %>%
  dplyr::select(-test_count) %>%
  gather(key = "term", value = "value") %>%
  ggplot(aes(x = term, y = value)) +
  geom_boxplot() +
  labs(title = "Comparison of LDA and QDA",
```

```
       x = "Error rates for comparison",
       y = NULL)
```



The data and the boxplots show that there's little difference in the mean error rates for LDA and QDA during training and testing. It's hard to tell which performs better at this level where n=1000.

Question 3

question 3 goes through basically the same process as question 2

```
repeat_test3 <- function(x){
  set.seed(x)
  sample1000 <- tibble(X1 = runif(1000, -1, 1),
                       X2 = runif(1000, -1, 1),
                       e = rnorm(1000, 0, 1))
  sample1000_split <- initial_split(sample1000, prop = 0.7)
  train_3 <- training(sample1000_split) %>%
    mutate(Y = X1 + X1^2 + X2 + X2^2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  test_3 <- testing(sample1000_split) %>%
    mutate(Y = X1 + X1^2 + X2 + X2^2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  lda3 <- lda(result ~ X1 + X1^2 + X2 + X2^2, data = train_3)
  qda3 <- qda(result ~ X1 + X1^2 + X2 + X2^2, data = train_3)
  ldapred_train3 <- predict(lda3, newdata = train_3)
```

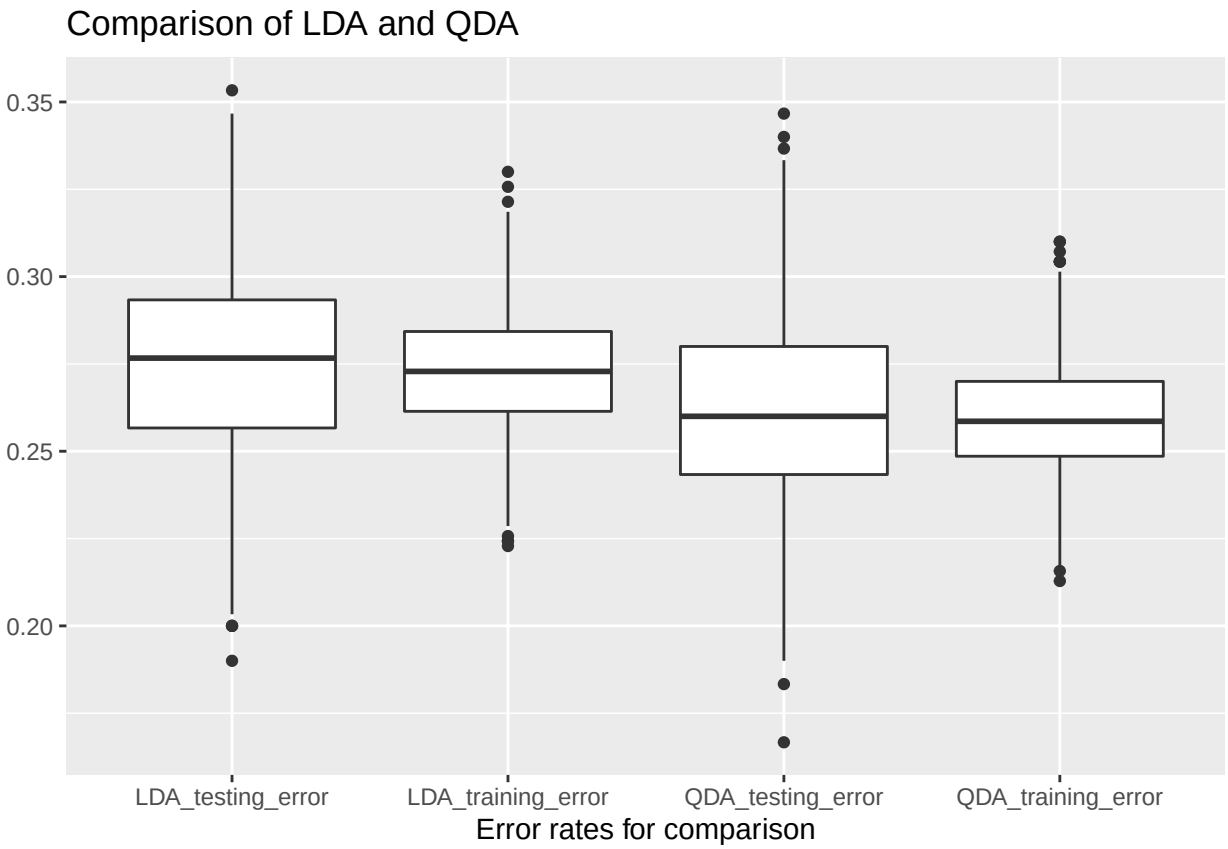
```
  ldapred_test3 <- predict(lda3, newdata = test_3)
  lda_train_error3 <- mean(train_3$result != ldapred_train3$class)
  lda_test_error3 <- mean(test_3$result != ldapred_test3$class)
  qdapred_train3 <- predict(qda3, newdata = train_3)
  qdapred_test3 <- predict(qda3, newdata = test_3)
  qda_train_error3 <- mean(train_3$result != qdapred_train3$class)
  qda_test_error3 <- mean(test_3$result != qdapred_test3$class)
  results3 <- data.frame("test_count" = as.factor(x),
            "LDA_training_error"  = lda_train_error3,
            "LDA_testing_error" = lda_test_error3,
            "QDA_training_error"  = qda_train_error3,
            "QDA_testing_error" = qda_test_error3)
  results3
}
```

```
results1000_3 <- map(1:1000, repeat_test3) %>%
  bind_rows()%>%
  dplyr::select(-test_count)
```

```
results1000_3 %>%
  summary() %>%
  as.table()
```

```
##  LDA_training_error LDA_testing_error QDA_training_error QDA_testing_error
##  Min.   :0.223      Min.   :0.190     Min.   :0.213      Min.   :0.167
##  1st Qu.:0.261      1st Qu.:0.257     1st Qu.:0.249      1st Qu.:0.243
##  Median :0.273      Median :0.277     Median :0.259      Median :0.260
##  Mean   :0.273      Mean   :0.276     Mean   :0.259      Mean   :0.262
##  3rd Qu.:0.284      3rd Qu.:0.293     3rd Qu.:0.270      3rd Qu.:0.280
##  Max.   :0.330      Max.   :0.353     Max.   :0.310      Max.   :0.347
```

```
results1000_3 %>%
  gather(key = "term", value = "value") %>%
  ggplot(aes(x = term, y = value)) +
  geom_boxplot() +
  labs(title = "Comparison of LDA and QDA",
       x = "Error rates for comparison",
       y = NULL)
```



The data and the boxplots show that when applied on non-linear relations, QDA tend to have a lower mean error rate than LDA, in both training and testing, while both models have similar degrees of variance in error rates, no matter in training or in testing. Therefore, QDA models perform better than LDA in both training and testing non-linear Bayesian decision boundaries.

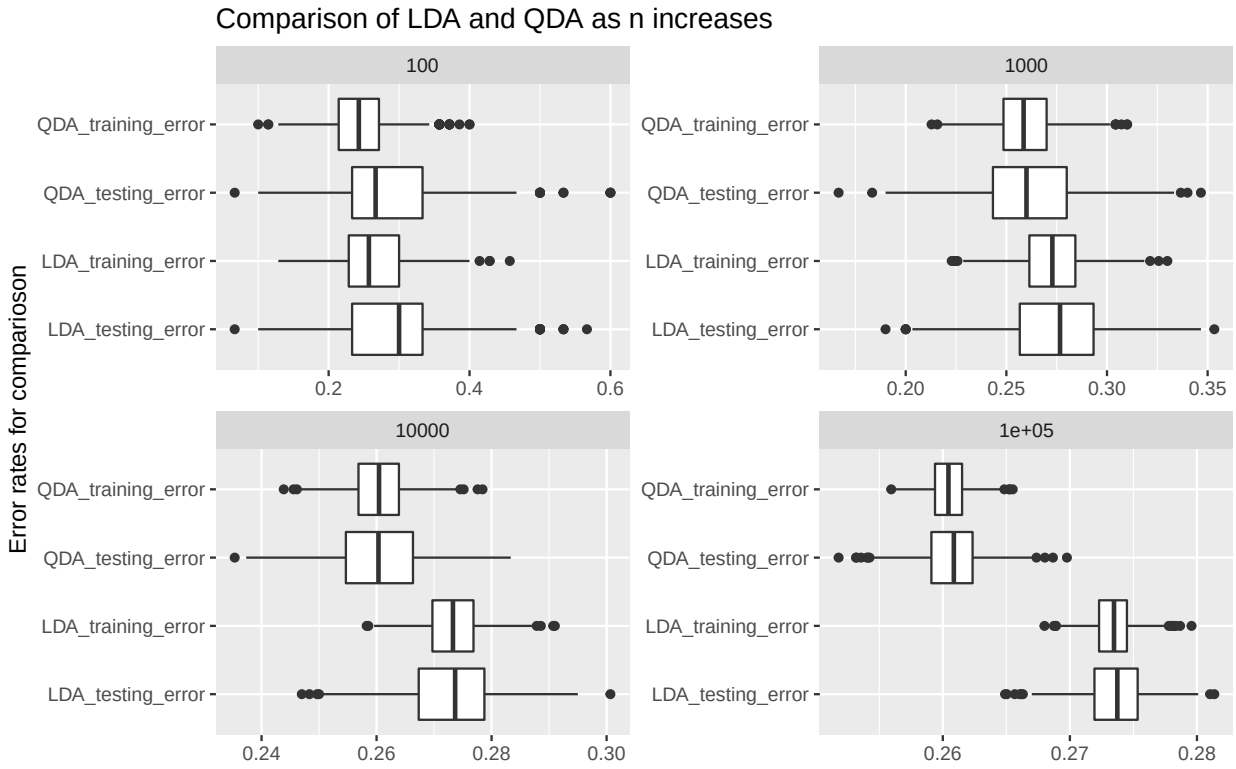
Question 4

```
#adds y into the function to change the size of n
repeat_test4 <- function(x, y){
  set.seed(x)
  sample <- tibble(X1 = runif(y, -1, 1),
                   X2 = runif(y, -1, 1),
                   e = rnorm(y, 0, 1))
  sample_split <- initial_split(sample, prop = 0.7)
  train_4 <- training(sample_split) %>%
    mutate(Y = X1 + X1^2 + X2 + X2^2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  test_4 <- testing(sample_split) %>%
    mutate(Y = X1 + X1^2 + X2 + X2^2 + e) %>%
    mutate(result = ifelse(Y >= 0, "TRUE", "FALSE"))
  lda4 <- lda(result ~ X1 + X1^2 + X2 + X2^2, data = train_4)
  qda4 <- qda(result ~ X1 + X1^2 + X2 + X2^2, data = train_4)
  ldapred_train4 <- predict(lda4, newdata = train_4)
  ldapred_test4 <- predict(lda4, newdata = test_4)
  lda_train_error4 <- mean(train_4$result != ldapred_train4$class)
```

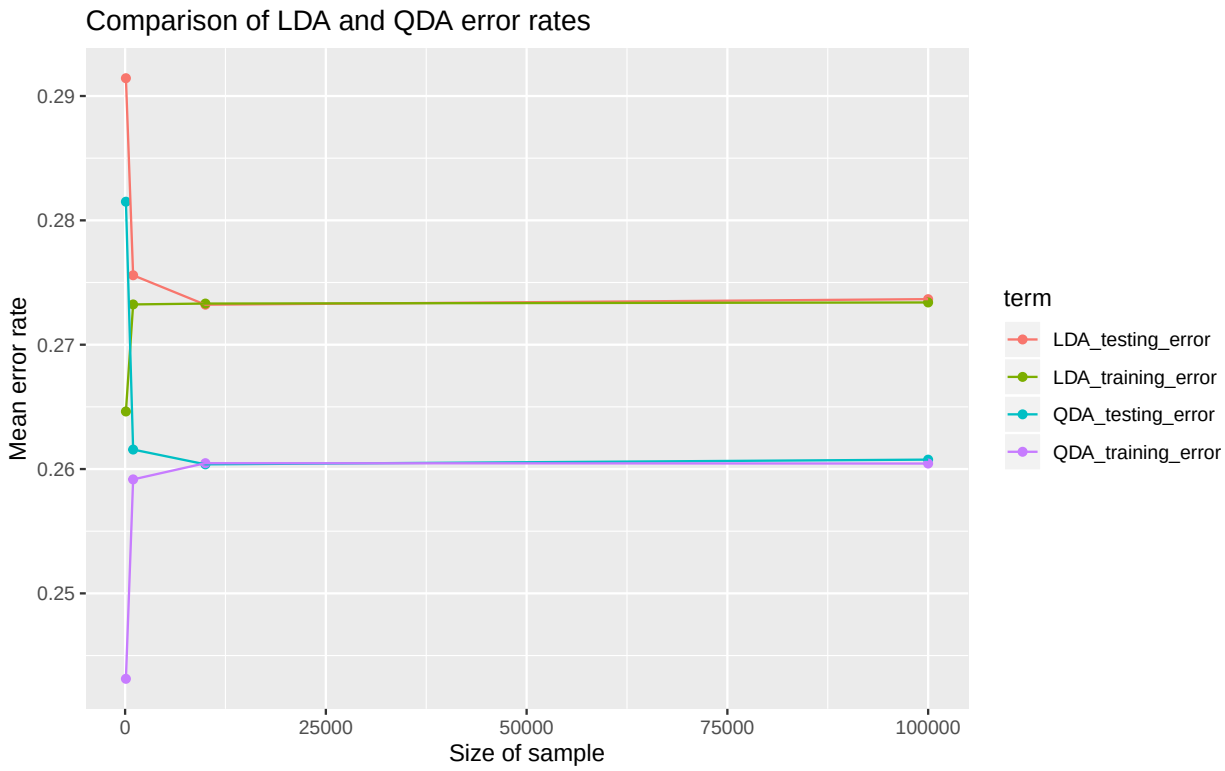
```
  lda_test_error4 <- mean(test_4$result != ldapred_test4$class)
  qdapred_train4 <- predict(qda4, newdata = train_4)
  qdapred_test4 <- predict(qda4, newdata = test_4)
  qda_train_error4 <- mean(train_4$result != qdapred_train4$class)
  qda_test_error4 <- mean(test_4$result != qdapred_test4$class)
  results4 <- data.frame("test_count" = as.factor(x),
                         "n_size" = y,
                         "LDA_training_error"  = lda_train_error4,
                         "LDA_testing_error" = lda_test_error4,
                         "QDA_training_error"  = qda_train_error4,
                         "QDA_testing_error" = qda_test_error4)
  results4
}
```

```
#the mapping process takes a very long time for some reason
xy_list <- expand_grid(x = 1:1000, y = c(1e02, 1e03, 1e04, 1e05))
results1000_4 <- map2(xy_list$x, xy_list$y, repeat_test4) %>%
  bind_rows()%>%
  dplyr::select(-test_count)
```

```
results1000_4 %>%
  gather(term, value, -n_size) %>%
  ggplot(aes(x = term, y = value)) +
  geom_boxplot() +
  scale_y_continuous() +
  coord_flip() +
  facet_wrap(~n_size, scales = "free") +
  labs(title = "Comparison of LDA and QDA as n increases",
       x = "Error rates for comparioson",
       y = NULL)
```




```
results1000_4 %>%
  gather(term, value, -n_size) %>%
  group_by(n_size, term) %>%
  summarize(mean = mean(value)) %>%
  ggplot(aes(x = n_size, y = mean, color = term)) +
  geom_point() +
  geom_line() +
  labs(title = "Comparison of LDA and QDA error rates",
       x = "Size of sample",
       y = "Mean error rate")
```



- As the size of n increases, both QDA and LDA will have lower test error rates, and both will have its mean training error rate and mean test error rate converge. However, both the boxplots and the line graph show that as n increases, the mean test error rate of QDA will become lower than LDA.
- It should be noted that in the example, there are only two independent variables. Even n increases significantly, the variance in the data may be relatively limited.
- The results agree with the idea that LDA is better fit for samples with smaller n, and that QDA is better fit for samples with larger n.

Modeling voter turnout

Question 5

```
GSS98 <- read_csv("mental_health.csv")%>%
  drop_na() %>%
  as.tibble()
```

```
## Parsed with column specification:
## cols(
##   vote96 = col_double(),
##   mhealth_sum = col_double(),
##   age = col_double(),
##   educ = col_double(),
##   black = col_double(),
##   female = col_double(),
##   married = col_double(),
```

```
##   inc10 = col_double()
## )
```

```
GSS98_split <- initial_split(GSS98, 0.7)
GSS98_train <- training(GSS98_split)
GSS98_test <- testing(GSS98_split)
```

```
#There were issues using training function here, so use lda and qda functions directly
#LDA model
lda5 <- lda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10,
            data = GSS98_train,
            family = binomial)

#QDA model
qda5 <- qda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10,
            data = GSS98_train,
            family = binomial)

#prepare data for training
GSS98_train_nb <- GSS98_train %>%
  mutate(vote96 = ifelse(vote96 == 1, "YES", "NO"))
X5 <- GSS98_train_nb %>%
  dplyr::select(mhealth_sum, age, educ, black, female, married, inc10)
Y5 <- GSS98_train_nb$vote96

#naive Bayesian model
nb5 <- train(
  x = X5,
  y = Y5,
  method = "nb"
)

#logistical regression model
glm5 <- train(
  x = X5,
  y = Y5,
  method = "glm",
)

#K-nn
knn_fun <- function(x){
  knn(GSS98_train, GSS98_test, GSS98_train$vote96, k = x)
}
knnpred <- map(1:10, knn_fun)
knn_result <- knn(GSS98_train, GSS98_test, GSS98_train$vote96, k = 1)
knn_result
```

```
##   [1] 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0
##  [38] 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 0 1 0 0 0 1 0 1 1 1 0 1 1
##  [75] 0 1 0 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1
## [112] 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1
## [149] 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0
## [186] 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0
```

```
## [223] 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1
## [260] 1 1 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0
## [297] 1 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 0
## [334] 0 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1
## Levels: 0 1
```

```
GSS98_test_nb <- GSS98_test %>%
  mutate(vote96 = ifelse(vote96 == 1, "YES", "NO"))

#error rate of glm model
glmpred5 <- predict(glm5, newdata = GSS98_test)
glm_count <- table(GSS98_test_nb$vote96 == glmpred5)
glm_rate <- glm_count[names(glm_count) == "FALSE"] / 349
glm_rate
```

```
## FALSE
## 0.275
```

```
#error rate of naive Bayesian model
nbpred5 <- predict(nb5, newdata = GSS98_test)
nb_count <- table(GSS98_test_nb$vote96 == nbpred5)
nb_rate <- nb_count[names(nb_count) == "FALSE"] / 349
nb_rate
```

```
## FALSE
## 0.304
```

```
#error rate of LDA model
ldapred5 <- predict(lda5, newdata = GSS98_test)
lda_count <- table(GSS98_test$vote96 == ldapred5$class)
lda_rate <- lda_count[names(lda_count) == "FALSE"] / 349
lda_rate
```

```
## FALSE
## 0.278
```

```
#error rate of QDA model
qdapred5 <- predict(qda5, newdata = GSS98_test)
qda_count <- table(GSS98_test$vote96 == qdapred5$class)
qda_rate <- qda_count[names(qda_count) == "FALSE"] / 349
qda_rate
```

```
## FALSE
## 0.309
```

```
#K-nn error rates
K <- 1:10
knn_rate <- function(x){
  knn_count <- table(GSS98_test$vote96 == knnpred[[x]])
  knn_err <- (knn_count[names(knn_count) == "FALSE"] / 349)
  knn_err
```

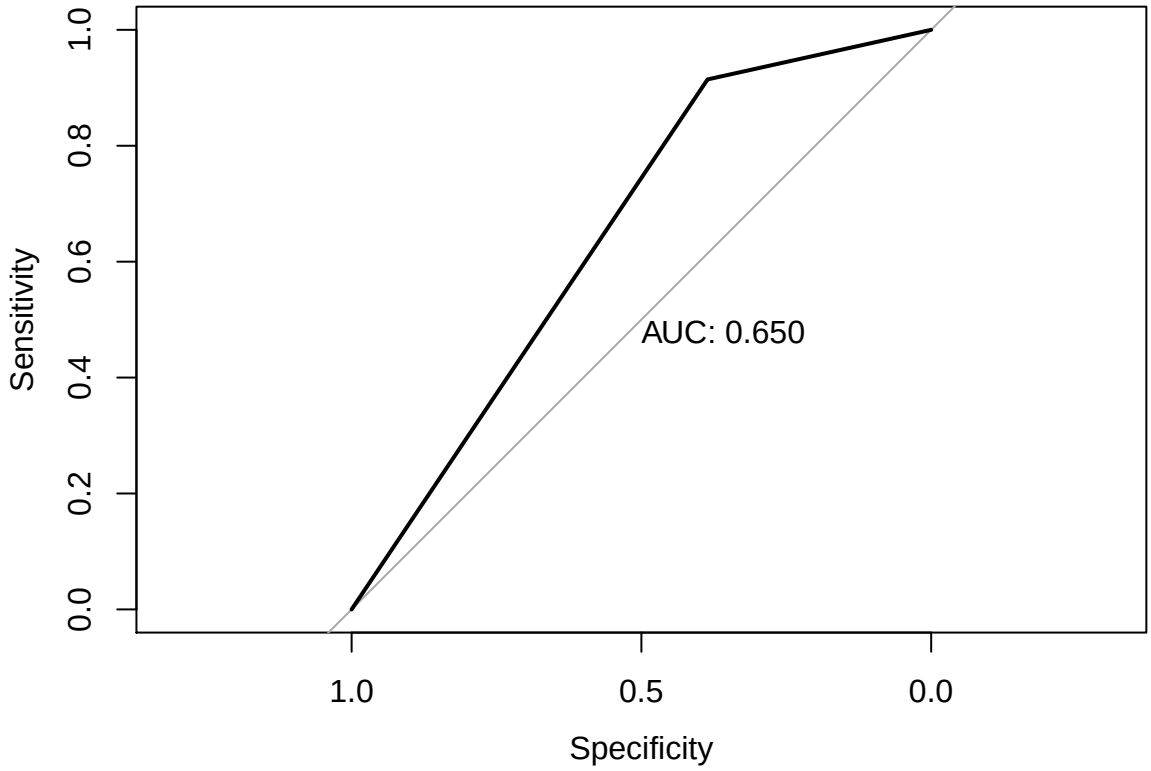
```
}
knn_result <- map(K, knn_rate)
knn_result
```

```
## [[1]]
## FALSE
## 0.241
##
## [[2]]
## FALSE
## 0.295
##
## [[3]]
## FALSE
## 0.264
##
## [[4]]
## FALSE
## 0.266
##
## [[5]]
## FALSE
## 0.264
##
## [[6]]
## FALSE
## 0.281
##
## [[7]]
## FALSE
## 0.278
##
## [[8]]
## FALSE
## 0.269
##
## [[9]]
## FALSE
## 0.278
##
## [[10]]
## FALSE
## 0.284
```

As the data show, in terms of error rates, LDA > glm > QDA = nb > K-nn(K>=5) The lowest error rate is in K-nn when K equals 10. Therefore, judging by error rates, K-nn performs the best among the model, given K equals 10.

```
#LDA
roc(response = GSS98_test$vote96,
                predictor =  as.numeric(ldapred5$class),
                plot = TRUE,
                print.auc=TRUE)
```

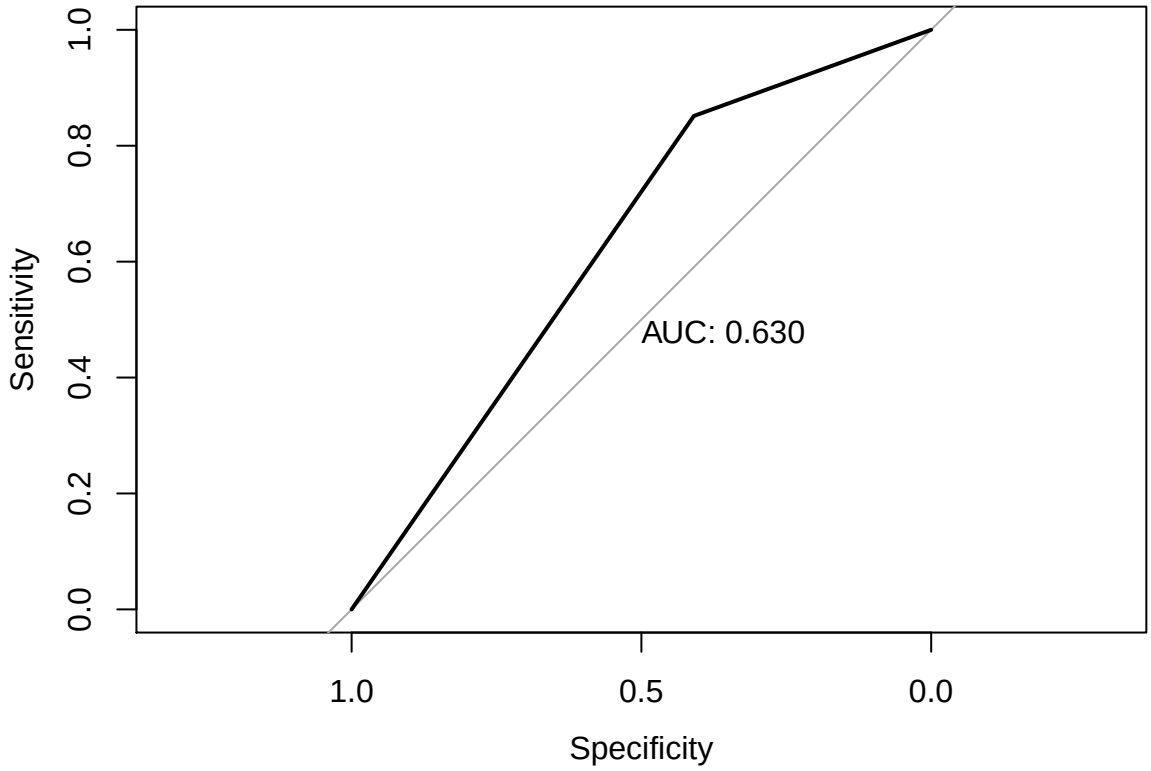
```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(ldapred5$class),     plot = TRUE, p
##
## Data: as.numeric(ldapred5$class) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote9
## Area under the curve: 0.65
```

```
#QDA
roc(response = GSS98_test$vote96,
               predictor =  as.numeric(qdapred5$class),
               plot = TRUE,
               print.auc=TRUE)
```

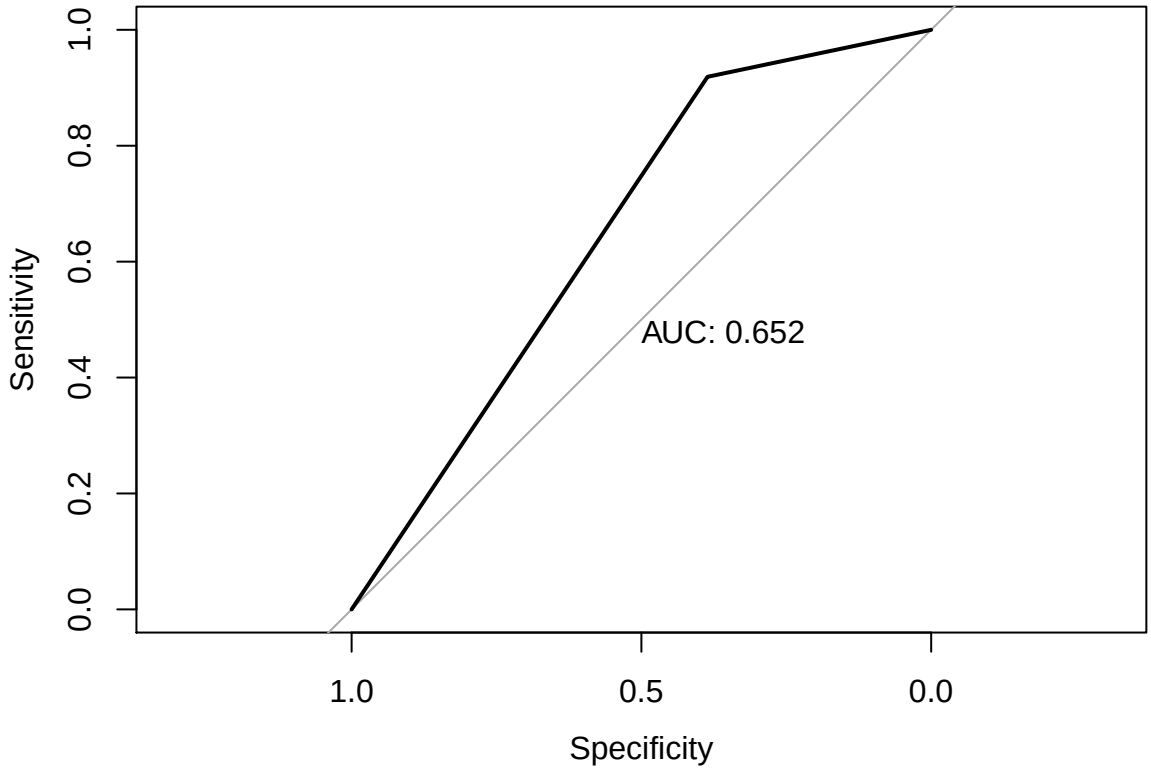
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(qdapred5$class),     plot = TRUE, pr
##
## Data: as.numeric(qdapred5$class) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96
## Area under the curve: 0.63
```

```
#glm
roc(response = GSS98_test_nb$vote96,
               predictor =  as.numeric(glmpred5),
               plot = TRUE,
               print.auc=TRUE)
```

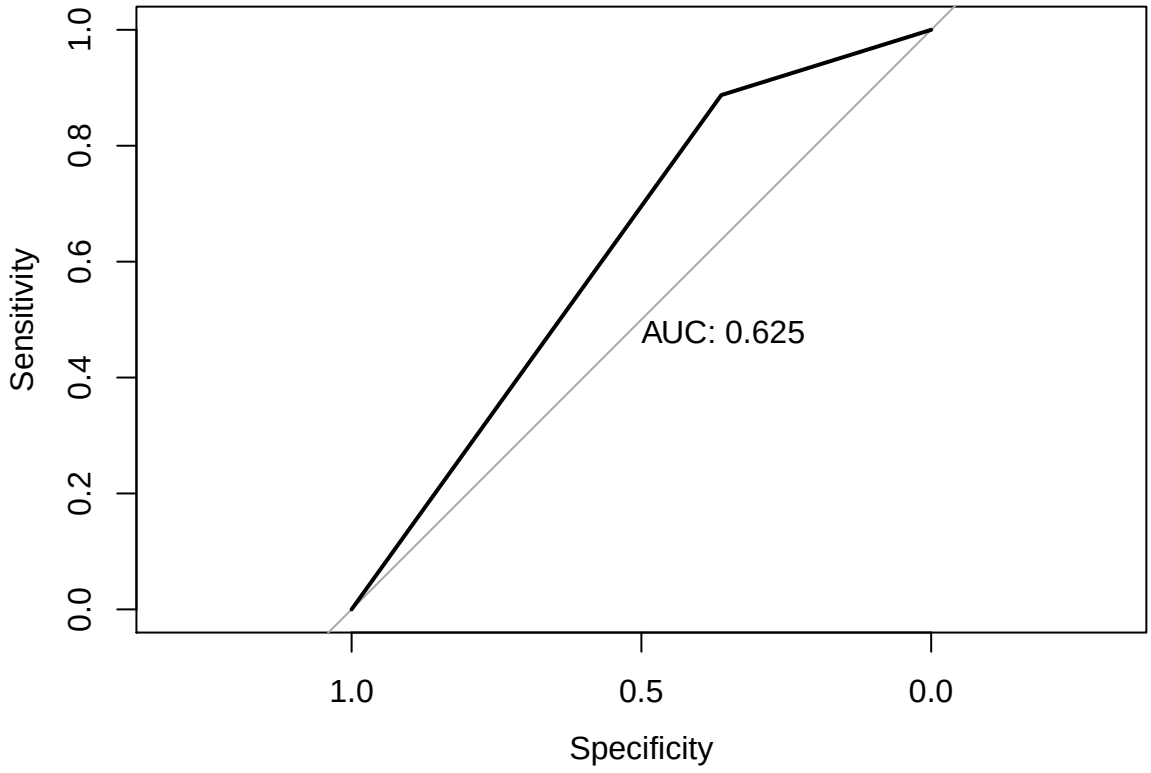
```
## Setting levels: control = NO, case = YES
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = GSS98_test_nb$vote96, predictor = as.numeric(glmpred5),     plot = TRUE, prin
##
## Data: as.numeric(glmpred5) in 127 controls (GSS98_test_nb$vote96 NO) < 222 cases (GSS98_test_nb$vote
## Area under the curve: 0.652
```

```
#naive Bayesian
roc(response = GSS98_test_nb$vote96,
               predictor =  as.numeric(nbpred5),
               plot = TRUE,
               print.auc=TRUE)
```

```
## Setting levels: control = NO, case = YES
## Setting direction: controls < cases
```



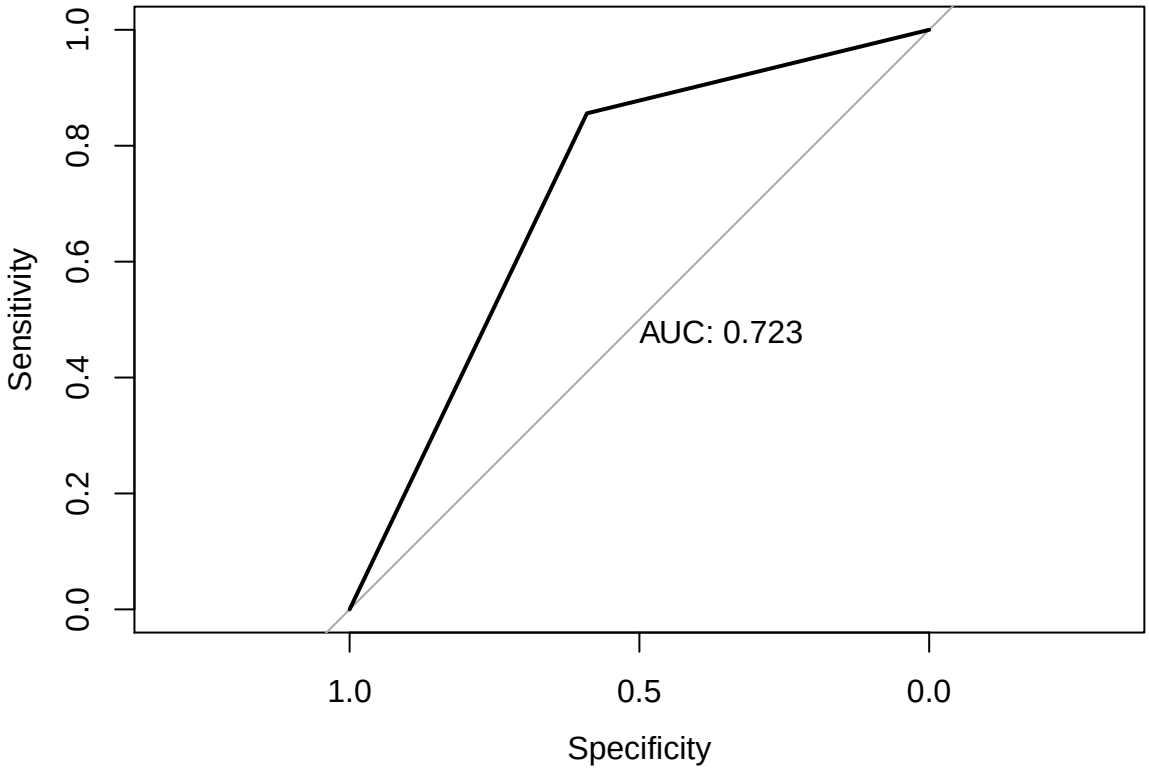

```
##
## Call:
## roc.default(response = GSS98_test_nb$vote96, predictor = as.numeric(nbpred5),     plot = TRUE, print
##
## Data: as.numeric(nbpred5) in 127 controls (GSS98_test_nb$vote96 NO) < 222 cases (GSS98_test_nb$vote9
## Area under the curve: 0.625
```

```
#K-nn
knn_roc_plot <- function(x){
  roc(response = GSS98_test$vote96,
              predictor =  as.numeric(knnpred[[x]]),
              plot = TRUE,
              print.auc=TRUE)
  }
map(K, knn_roc_plot)
```

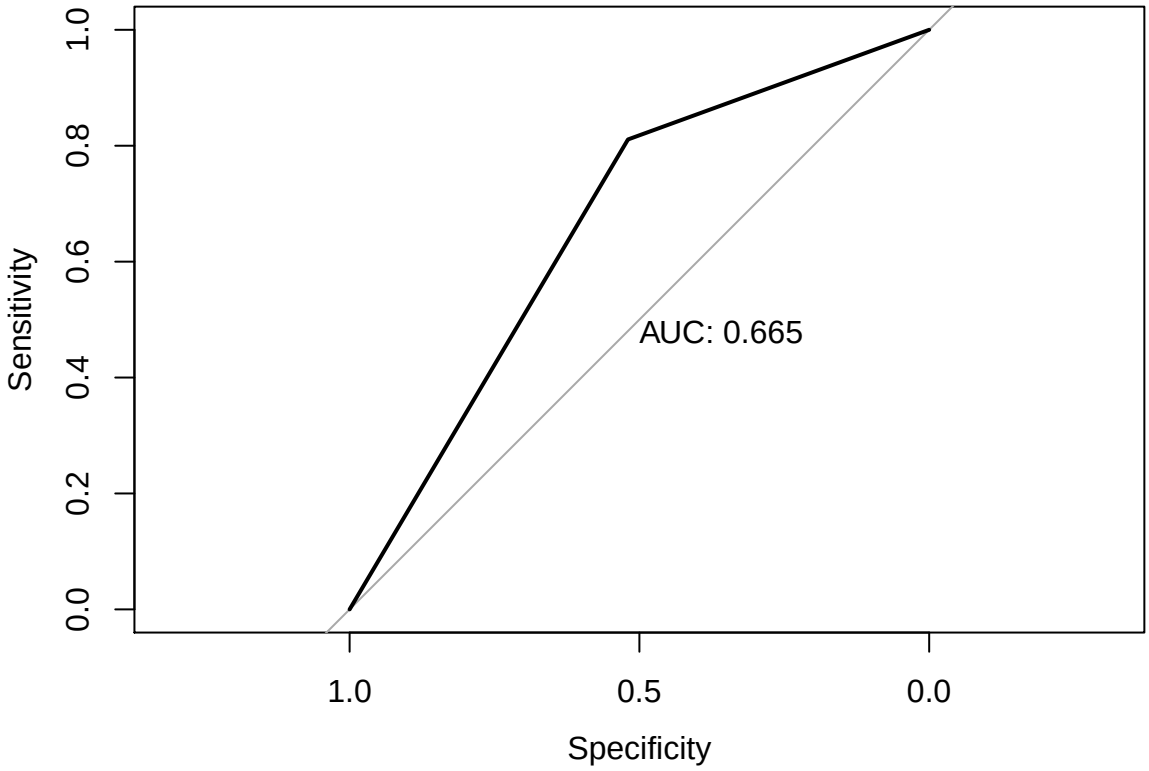
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
## Setting levels: control = 0, case = 1
```

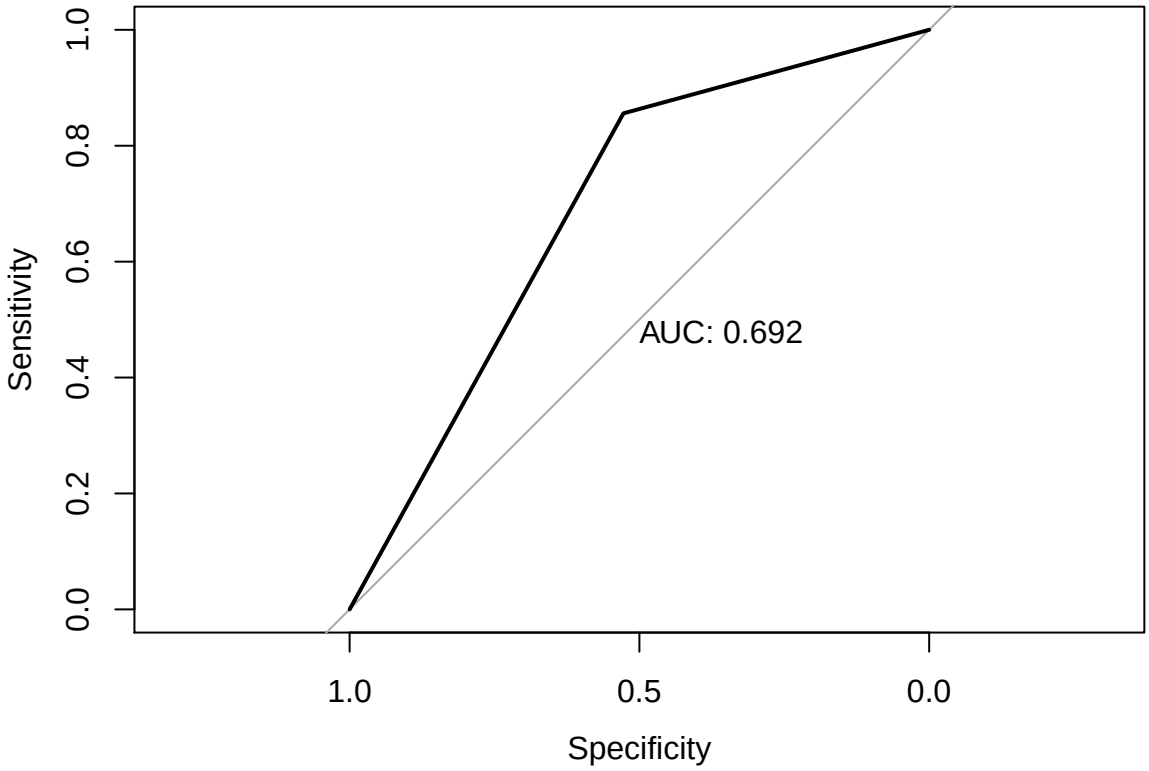
```
## Setting direction: controls < cases
```



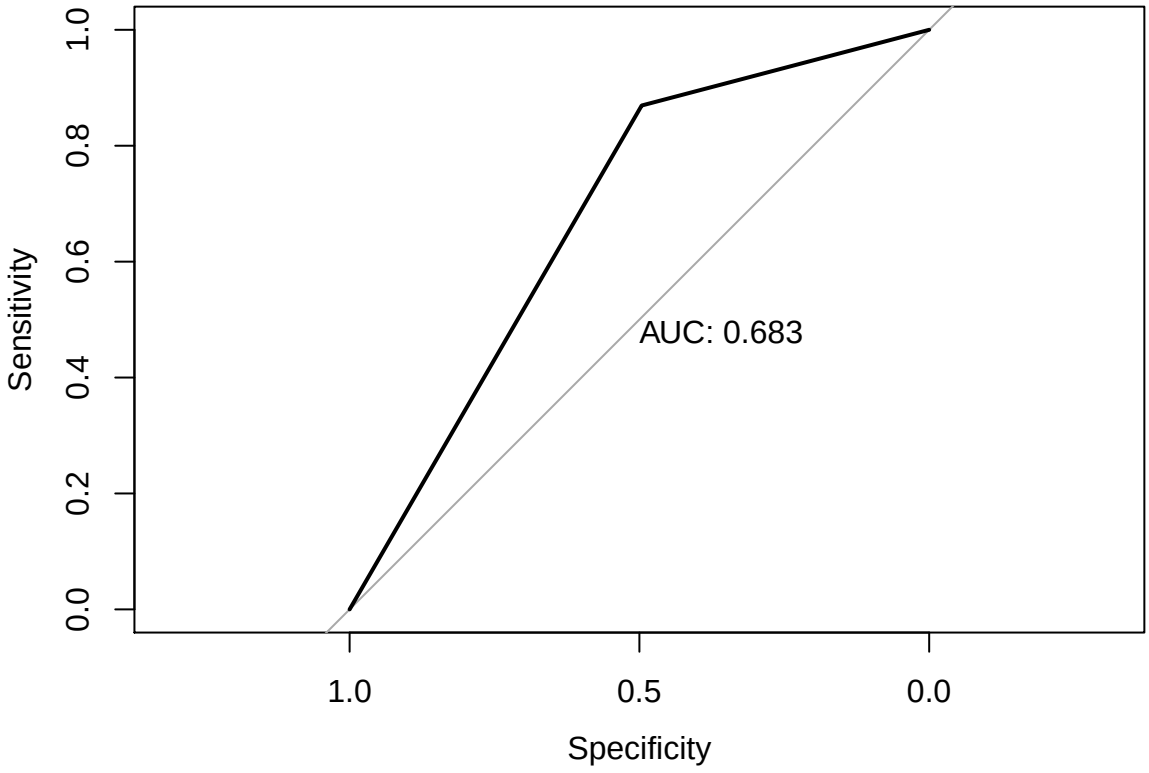

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



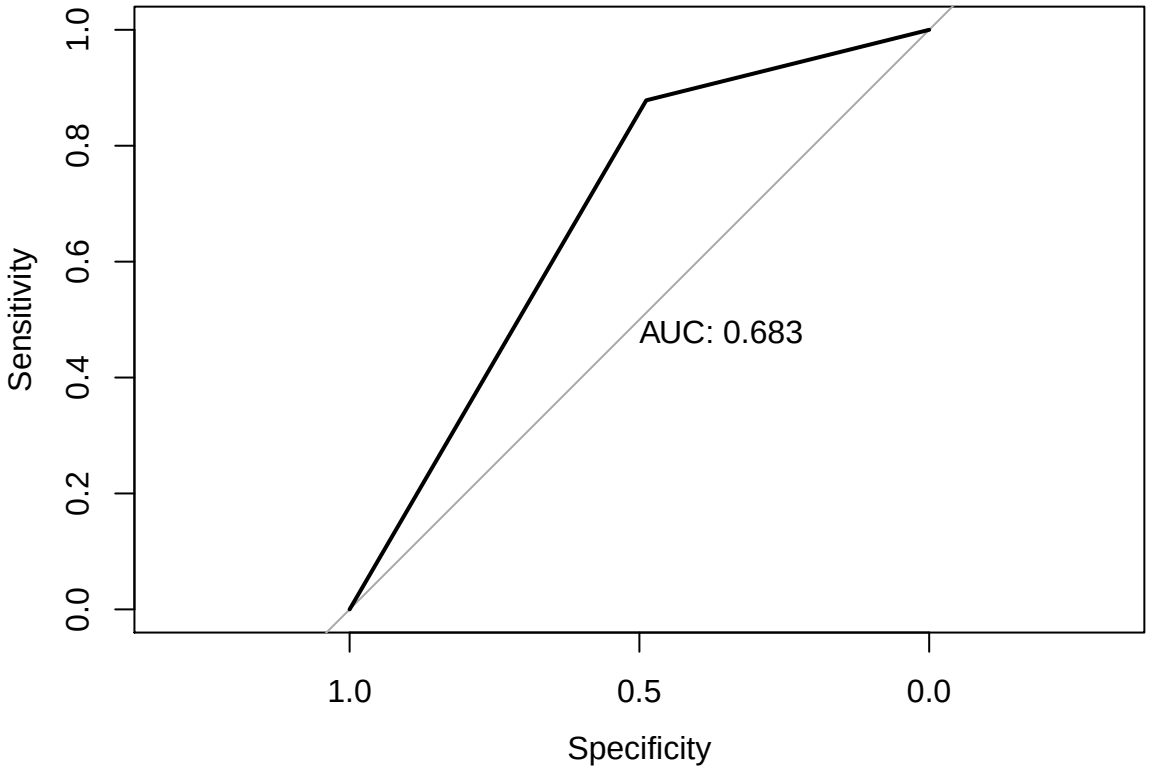

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



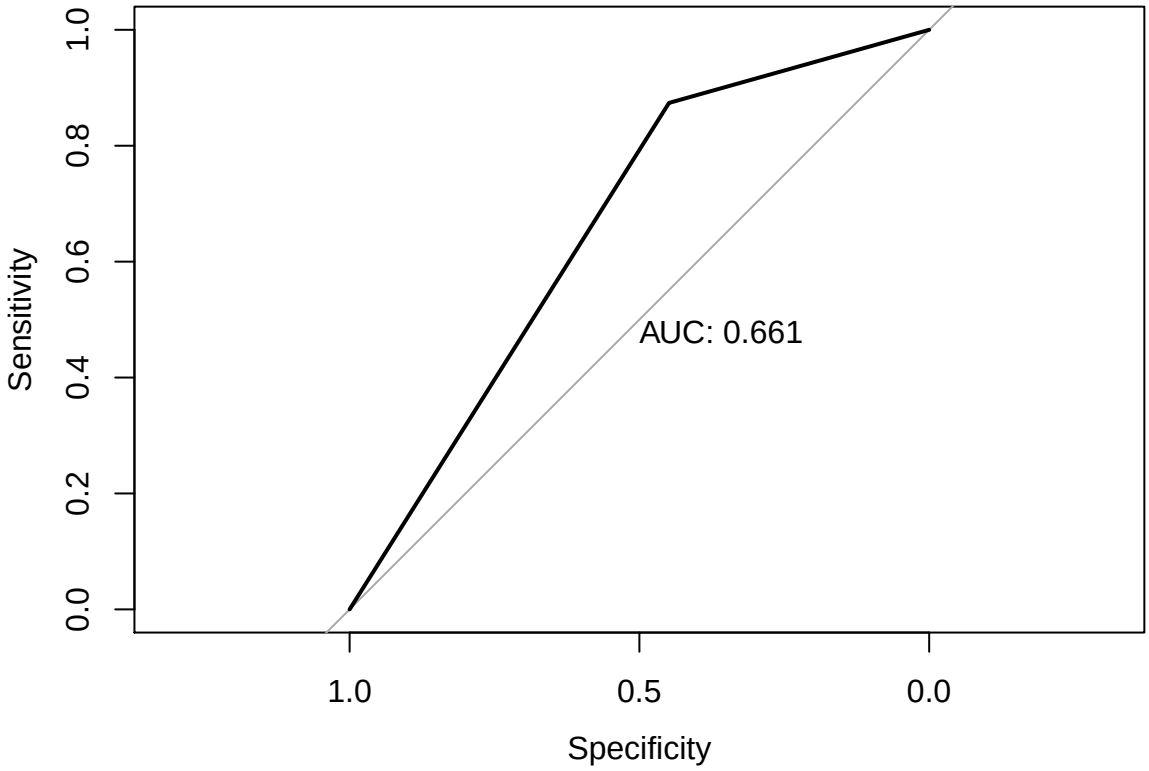

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



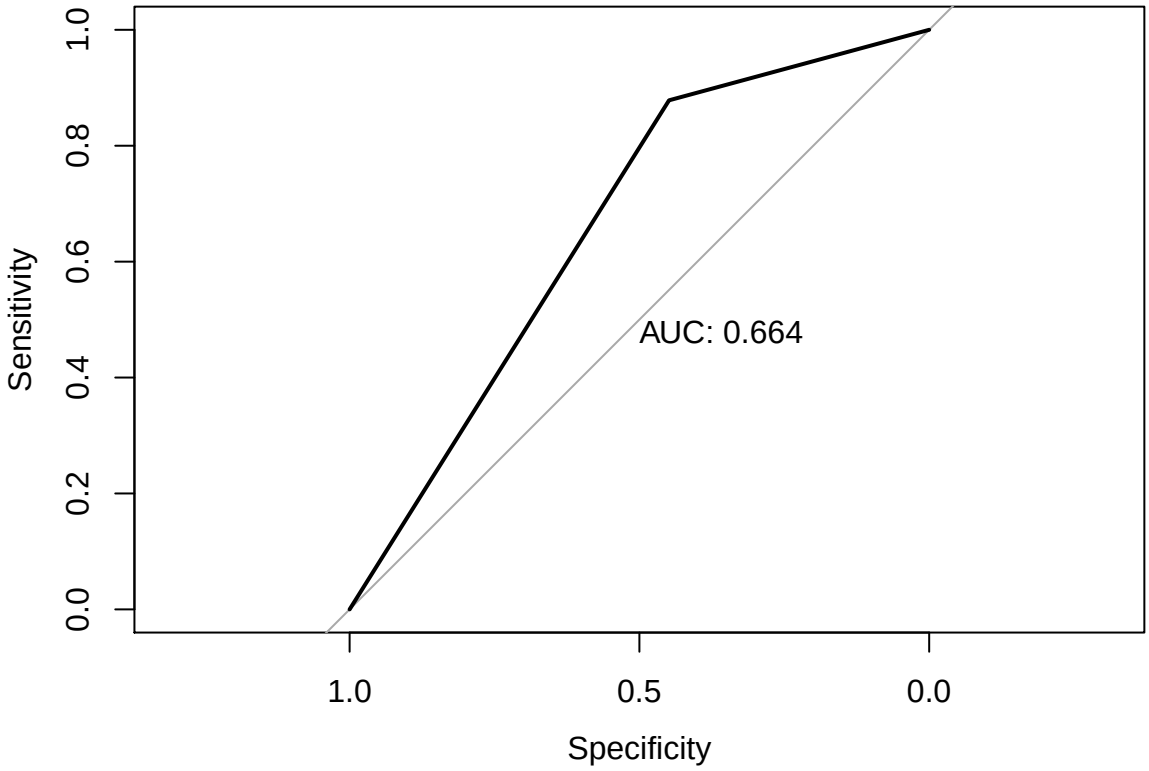

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



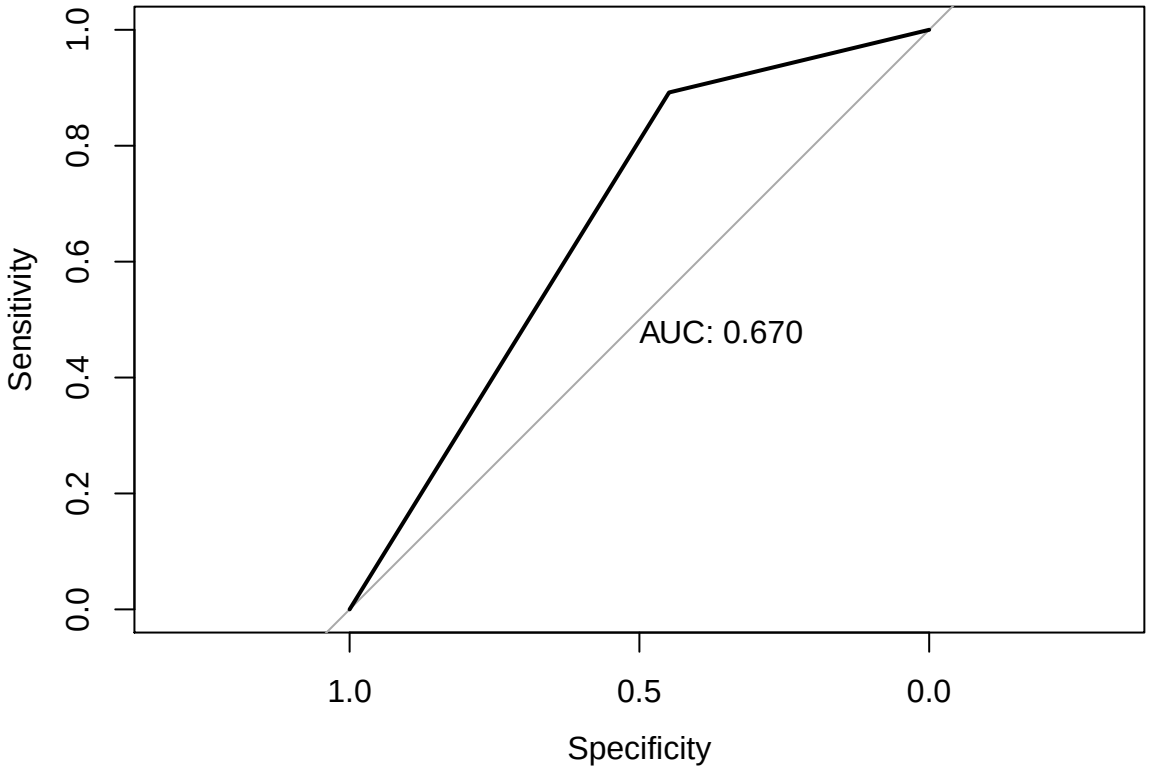

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



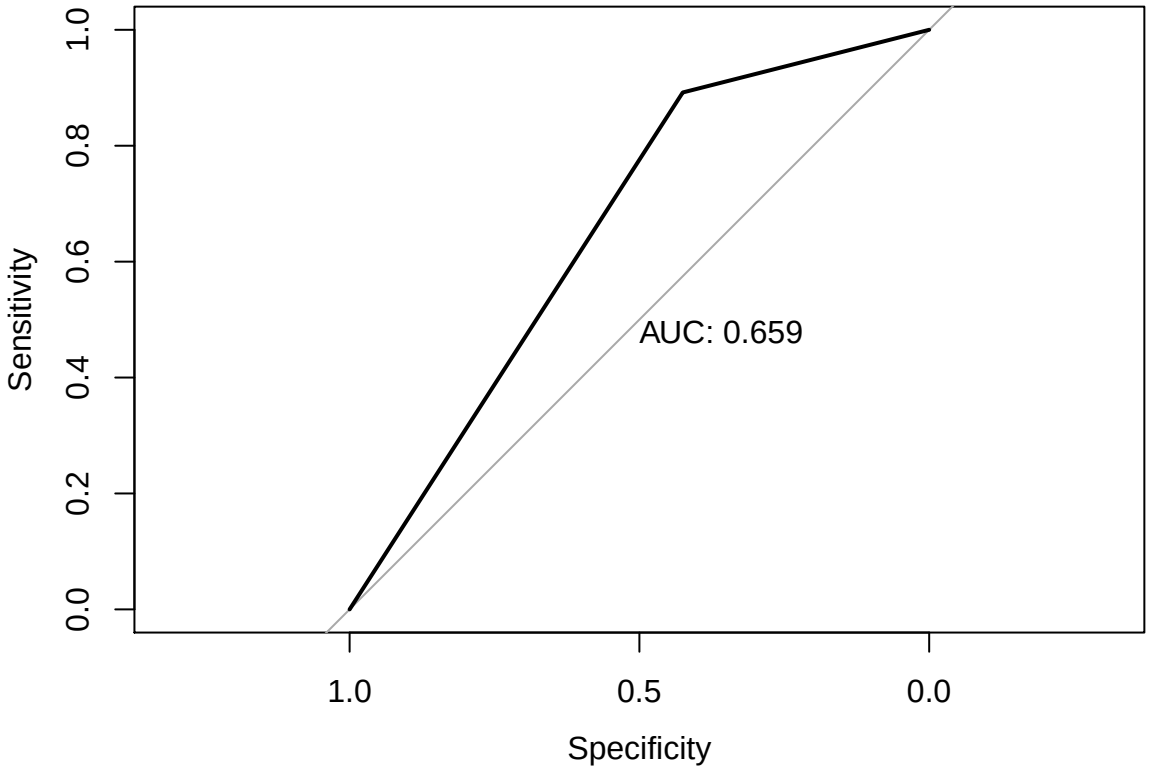

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

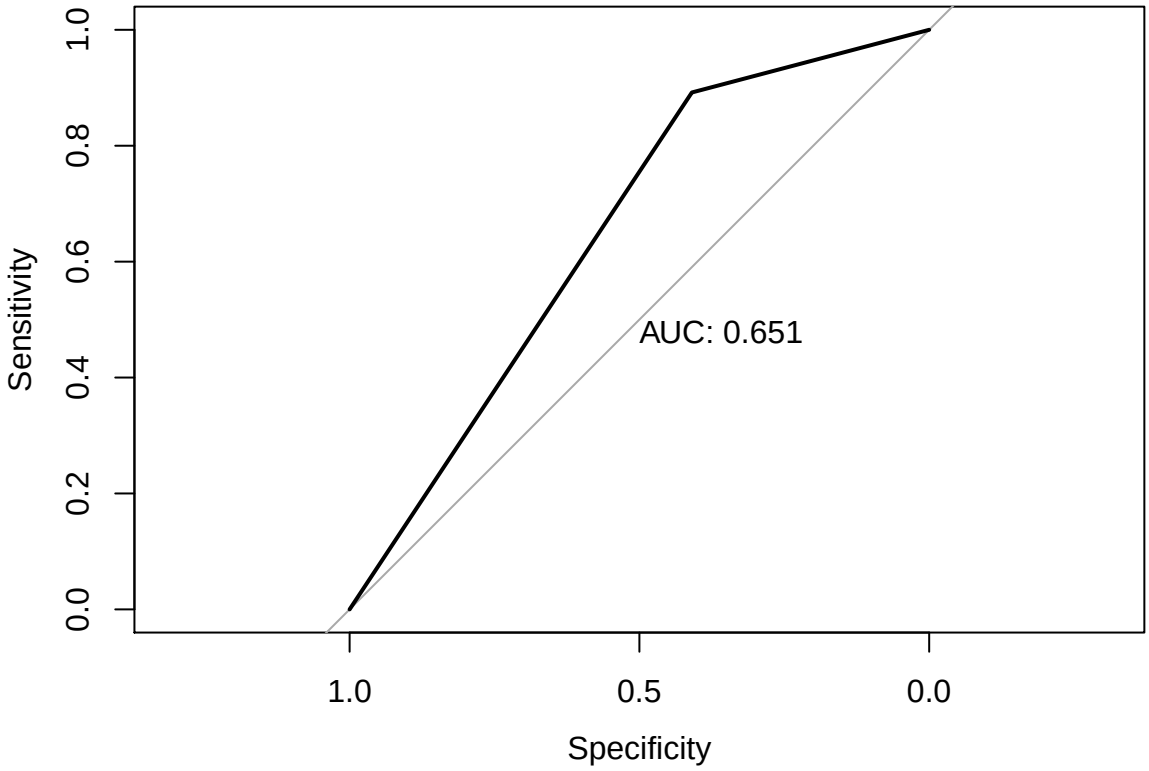



```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```


Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
AUC: 0.659
1.0
0.5
0.0
Specificity




```
## [[1]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 
## Area under the curve: 0.723
##
## [[2]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 
## Area under the curve: 0.665
##
## [[3]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 
## Area under the curve: 0.692
##
## [[4]]
##
```

```
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.683
##
## [[5]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.683
##
## [[6]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.661
##
## [[7]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.664
##
## [[8]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.67
##
## [[9]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.659
##
## [[10]]
##
## Call:
## roc.default(response = GSS98_test$vote96, predictor = as.numeric(knnpred[[x]]),     plot = TRUE, pri
##
## Data: as.numeric(knnpred[[x]]) in 127 controls (GSS98_test$vote96 0) < 222 cases (GSS98_test$vote96 1
## Area under the curve: 0.651
```

However, calculation of ROC and AUC shows that when K equals 9, AUC reaches maximum, and that when K equals 7 or 9, AUC is greater than when K equals 10. There may be errors due to how AUC is calculated by the function roc(), or other unknown factors. Nevertheless, the conclusion will stick to the results of error rates. The best model is still K-nn when K is 10.