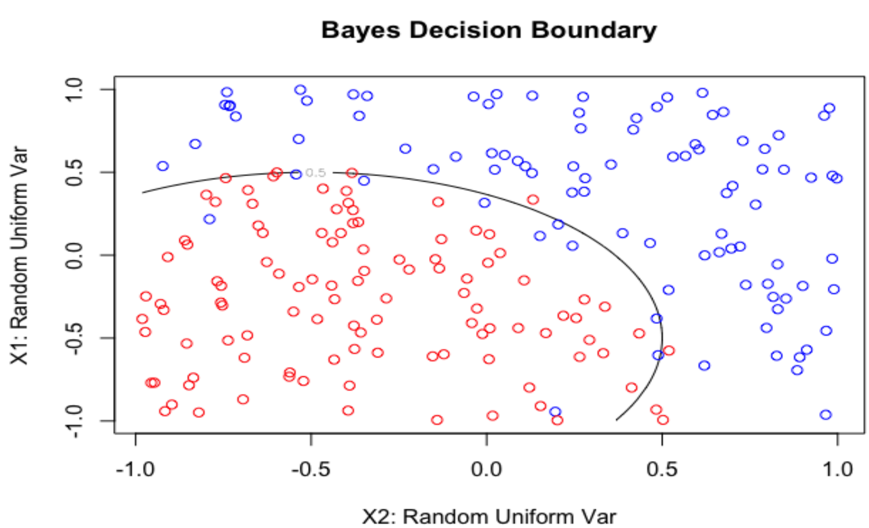
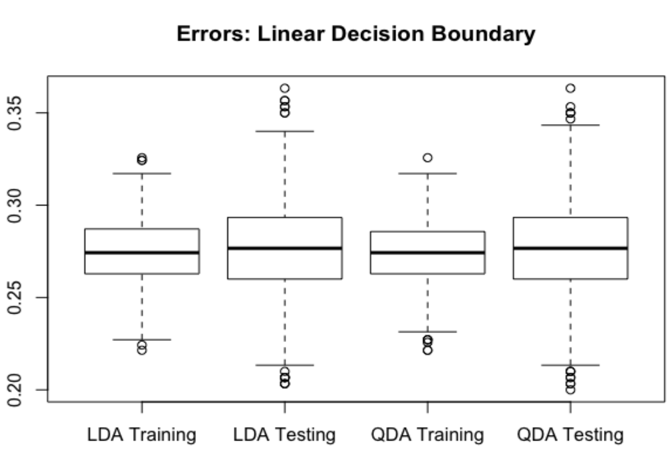
```
# Assignment 2

# Load in packages
library(rsample)
library(tables)
library(reshape2)
library(e1071)


# Question 1
# (a, b, c, d, e)
set.seed(1234)

X1 <- runif(200, -1, 1)
X2 <- runif(200, -1, 1)
error <- rnorm(200, 0, 0.25)
Y <- X1 + X2 + X1^2 + X2^2 + error
prob <- exp(Y) / (1 + exp(Y))

X1_ordered = X1[order(X1)]
X2_ordered = X2[order(X2)]

Z <- expand.grid(X1_ordered, X2_ordered)
Z <- Z[1] + Z[2] + Z[1]^2 + Z[2]^2
Z <- data.matrix(Z)

# (f, g)
contour(X1_ordered, X2_ordered, matrix(Z, nrow=200), levels = 0.5,
     main = "Bayes Decision Boundary", ylab = "X1: Random Uniform Var",
     xlab = "X2: Random Uniform Var")
points(X1, X2, col = ifelse(Y < 0.5, 'red', 'blue'))
```



```
# Question 2
# (a)
iterations = 1000
errors <- matrix(ncol=4, nrow=iterations)
for (i in 1:iterations) {
  X1 <- runif(1000, -1, 1)
  X2 <- runif(1000, -1, 1)
  error <- rnorm(1000, 0, 1)
  Y_sim <- X1 + X2 + error
  Y_sim <- Y_sim >= 0

  df_model <- data.frame(X1, X2, Y_sim)

  split <- initial_split(df_model, prop=0.7)
  train <- training(split)
  test <- testing(split)

  lda.model <- MASS::lda(Y_sim ~ X1 + X2, data=train)
  ldapred.train <- predict(lda.model, train)$class
  ldapred.test <- predict(lda.model, test)$class

  qda.model <- MASS::qda(Y_sim ~ X1 + X2, data=train)
  qdapred.train <- predict(qda.model, train)$class
  qdapred.test <- predict(qda.model, test)$class

  errors[i, 1] <- mean(ldapred.train != train$Y_sim)
  errors[i, 2] <- mean(ldapred.test != test$Y_sim)
  errors[i, 3] <- mean(qdapred.train != train$Y_sim)
  errors[i, 4] <- mean(qdapred.test != test$Y_sim)
}

colnames(errors) <- c("LDA Training", "LDA Testing", "QDA Training", "QDA Testing")
boxplot(errors, main = "Errors: Linear Decision Boundary")
melted_errors <- melt(errors, id.vars = NULL)
summary(errors)
```

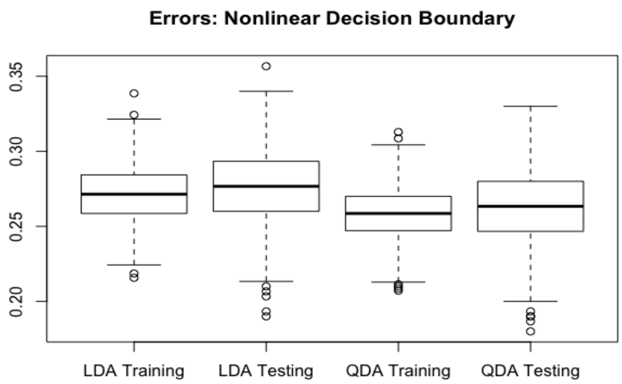



```
 LDA Training     LDA Testing      QDA Training     QDA Testing
Min.   :0.2214   Min.   :0.2033   Min.   :0.2214   Min.   :0.2000
1st Qu.:0.2629   1st Qu.:0.2600   1st Qu.:0.2629   1st Qu.:0.2600
Median :0.2743   Median :0.2767   Median :0.2743   Median :0.2767
Mean   :0.2751   Mean   :0.2768   Mean   :0.2745   Mean   :0.2772
3rd Qu.:0.2871   3rd Qu.:0.2933   3rd Qu.:0.2857   3rd Qu.:0.2933
Max.   :0.3257   Max.   :0.3633   Max.   :0.3257   Max.   :0.3633
```

(b) The boxplots indicate that the error rate differences between training and test across LDA and QDA are fairly similar. However, the tabular data show that LDA performs slightly better on the testing set because of the risk of QDA to overfit, while QDA performs better on the training set due to better model flexibility.

```r
# Question 3
# (a)
# Create function
discriminant_simulation <- function(N) {
  iterations = 1000
  errors <- matrix(ncol=4, nrow=iterations)
  for (i in 1:iterations) {
    X1 <- runif(N, -1, 1)
    X2 <- runif(N, -1, 1)
    error <- rnorm(N, 0, 1)
    Y_sim <- X1 + X1^2 + X2 + X2^2 + error
    Y_sim <- Y_sim >= 0

    df_model <- data.frame(X1, X2, Y_sim)

    split <- initial_split(df_model, prop=0.7)
    train <- training(split)
    test <- testing(split)

    lda.model <- MASS::lda(Y_sim ~ X1 + X1^2 + X2 + X2^2 , data=train)
    qda.model <- MASS::qda(Y_sim ~ X1 + X1^2 + X2 + X2^2 , data=train)
    ldapred.train <- predict(lda.model, train)$class
    qdapred.train <- predict(qda.model, train)$class
    ldapred.train <- predict(lda.model, test)$class
    qdapred.train <- predict(qda.model, test)$class


    errors[i, 1] <- mean(ldapred.train != train$Y_sim)
    errors[i, 2] <- mean(ldapred.test != test$Y_sim)
    errors[i, 3] <- mean(qdapred.train != train$Y_sim)
    errors[i, 4] <- mean(qdapred.test != test$Y_sim)
  }
  colnames(errors) <- c("LDA Training", "LDA Testing", "QDA Training", "QDA Testing")
  return(errors)
}
```

```
errors <- discriminant_simulation(1000)
boxplot(errors, main = "Errors: Nonlinear Decision Boundary")
melted_errors <- melt(errors, id.vars = NULL)
```

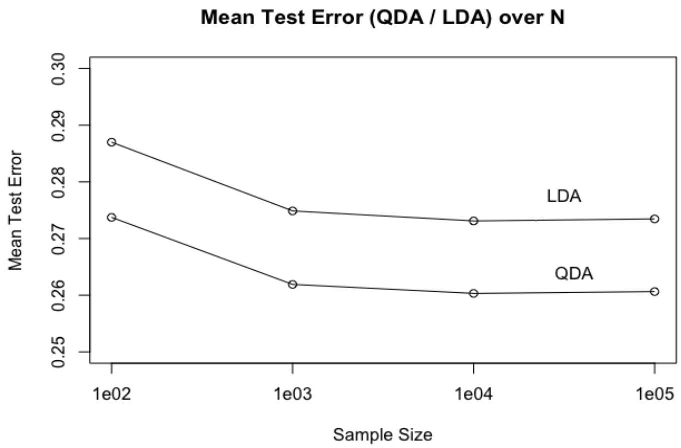



```
 LDA Training      LDA Testing       QDA Training      QDA Testing
Min.   :0.2200   Min.   :0.3314   Min.   :0.2100   Min.   :0.3443
1st Qu.:0.2614   1st Qu.:0.3911   1st Qu.:0.2471   1st Qu.:0.4029
Median :0.2729   Median :0.4043   Median :0.2593   Median :0.4186
Mean   :0.2728   Mean   :0.4056   Mean   :0.2588   Mean   :0.4186
3rd Qu.:0.2843   3rd Qu.:0.4214   3rd Qu.:0.2700   3rd Qu.:0.4343
Max.   :0.3257   Max.   :0.4943   Max.   :0.3114   Max.   :0.5057
```

(b) As before, QDA performs better than LDA on the training set. However, the nonlinear decision boundary captures nonlinearities in the data that are indeed uncaptured by LDA, so QDA outperforms LDA on the test set as well. Overall, QDA significantly outperforms LDA in the nonlinear case, which is to be expected based on the nonlinear model specification.

```
# Question 4
# (a)
# Different sizes
errors_1e02 <- discriminant_simulation(100)
errors_1e03 <- discriminant_simulation(1000)
errors_1e04 <- discriminant_simulation(10000)
errors_1e05 <- discriminant_simulation(100000)

nsize_LDA_test_errors <- cbind(errors_1e02[,"LDA Testing"], errors_1e03[,"LDA Testing"],
                               errors_1e04[,"LDA Testing"], errors_1e05[,"LDA Testing"])
nsize_QDA_test_errors <- cbind(errors_1e02[,"QDA Testing"], errors_1e03[,"QDA Testing"],
                               errors_1e04[,"QDA Testing"], errors_1e05[,"QDA Testing"])
n_labels = c("1e02", "1e03", "1e04", "1e05")
plot(colMeans(nsize_LDA_test_errors), type="o", xlab = "Sample Size",
   ylab="Mean Test Error", main="Mean Test Error (QDA / LDA) over N",
   ylim=c(0.4, 0.43), xaxt="n")
lines(colMeans(nsize_QDA_test_errors), type="o")
axis(1, at=1:4, labels=n_labels)
text(locator(), labels = c("LDA", "QDA"))

# Summaries
colnames(nsize_LDA_test_errors) = n_labels
colnames(nsize_QDA_test_errors) = n_labels
```

```
summary(nsize_LDA_test_errors)
summary(nsize_QDA_test_errors)
```



LDA Training	LDA Testing	QDA Training	QDA Testing
Min. :0.2214	Min. :0.2033	Min. :0.2214	Min. :0.2000
1st Qu.:0.2629	1st Qu.:0.2600	1st Qu.:0.2629	1st Qu.:0.2600
Median :0.2743	Median :0.2767	Median :0.2743	Median :0.2767
Mean :0.2751	Mean :0.2768	Mean :0.2745	Mean :0.2772
3rd Qu.:0.2871	3rd Qu.:0.2933	3rd Qu.:0.2857	3rd Qu.:0.2933
Max. :0.3257	Max. :0.3633	Max. :0.3257	Max. :0.3633

LDA Training	LDA Testing	QDA Training	QDA Testing
Min. :0.2200	Min. :0.3314	Min. :0.2100	Min. :0.3443
1st Qu.:0.2614	1st Qu.:0.3911	1st Qu.:0.2471	1st Qu.:0.4029
Median :0.2729	Median :0.4043	Median :0.2593	Median :0.4186
Mean :0.2728	Mean :0.4056	Mean :0.2588	Mean :0.4186
3rd Qu.:0.2843	3rd Qu.:0.4214	3rd Qu.:0.2700	3rd Qu.:0.4343
Max. :0.3257	Max. :0.4943	Max. :0.3114	Max. :0.5057

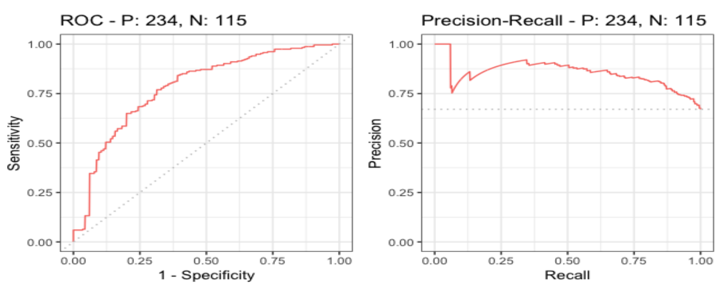
(b) As sample size increases, we expect the test error rate of both LDA and QDA to decline, and the results depict that. This is in contrast to what we would expect to happen with the training error rate as sample size grows. The intuition behind this is that as the model overcomes the overfitting problem with larger N, it performs better on the test data while sacrificing accuracy on the training data. It’s also important to note that although the LDA and QDA test error rates trend together, QDA consistently outperforms LDA. As before, this is because QDA is better at capturing the nonlinear decision boundary. Finally, the

```
# Question 5
# (a)
mental_df <- na.omit(read.csv("mental_health.csv")) %>%
  mutate(vote96 = as_factor(vote96),
         mhealth_sum = as.integer(mhealth_sum),
         age = as.integer(age),
         educ = as.integer(educ),
         black = as_factor(black),
         female = as_factor(female),
         married = as_factor(married))

mh_split <- initial_split(mental_df, prop=0.7)
mh_train <- training(mh_split)
mh_test <- testing(mh_split)

# (b)
# Logit
logit_model <- glm(vote96 ~ ., data = mh_train, family=binomial(link="logit"))
# LDA
lda_model <- MASS::lda(vote96 ~ ., data=mh_train)
# QDA
```

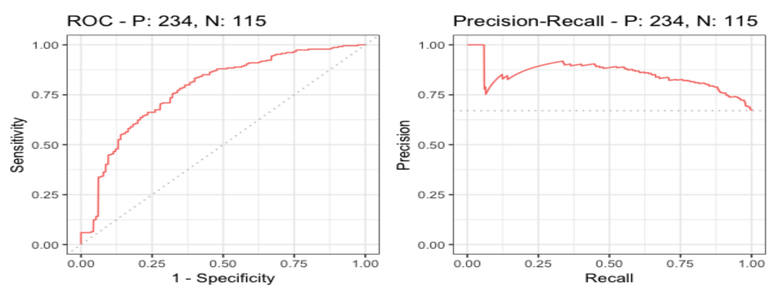
```r
qda_model <- MASS::qda(vote96 ~ ., data=mh_train)
# Naive Bayes
nbayes_model <- naiveBayes(vote96 ~ ., data=mh_train)
# K Means (best model is k = 10)
knn_model_errors <- tibble(k = 1:10,
                test_knn = map(k, ~ class::knn(train = select(mh_train, -vote96),
                                        test = select(mh_test, -vote96),
                                        cl = mh_train$vote96, k = .)),
                error = map_dbl(test_knn, ~ mean(mh_test$vote96 != .))) %>%
  mutate(model = paste0("k = ", k))


logit_pred <- exp(predict(logit_model, mh_test)) / (1 + exp(predict(logit_model, mh_test))
logit_pred_prob <- logit_pred > 0.5
lda_pred <- predict(lda_model, mh_test)
qda_pred <- predict(qda_model, mh_test)
nbayes_disc <- predict(nbayes_model, mh_test)
nbayes_pred <- predict(nbayes_model, mh_test, type="raw")[,2]

logit_err <- mean(logit_pred_prob != mh_test$vote96)
lda_err <- mean(lda_pred$class != mh_test$vote96)
qda_err <- mean(qda_pred$class != mh_test$vote96)
nbayes_err <- mean(nbayes_disc != mh_test$vote96)
knn_10_error <- knn_model_errors[10, 'error'][[1]]
test_errors <- cbind("Logit Error" = 0.2473265, "LDA Error" = lda_err, "QDA Error" = qda_err,
                "Naive Bayes Error" = nbayes_err, "KNN (k = 10) Best Error" = knn_10_error)

# ROC curves
logit_roc <- evalmod(scores = logit_pred, labels = mh_test$vote96)
lda_roc <- evalmod(scores = lda_pred$posterior[,2], labels = mh_test$vote96)
qda_roc <- evalmod(scores = qda_pred$posterior[,2], labels = mh_test$vote96)
nbayes_roc <- evalmod(scores = nbayes_pred, labels = mh_test$vote96)

autoplot(logit_roc)
text(locator(), labels = "Logit ROC")
autoplot(lda_roc)
text(locator(), labels = "LDA ROC")
autoplot(qda_roc)
text(locator(), labels = "QDA ROC")
autoplot(nbayes_roc)
text(locator(), labels = "Naive Bayes ROC")

# (c)
```

```r
# ROC for KNN (K = 10, best error rate (0.307))
# Source:
https://stackoverflow.com/questions/11741599/how-to-plot-a-roc-curve-for-a-knn-model
knn_10 <- class::knn(select(mh_train, -vote96),
              test = select(mh_test, -vote96),
              cl = mh_train$vote96, k = 10, prob=TRUE)
prob <- attr(knn_10, "prob")
prob <- 2*ifelse(knn_10 == "-1", 1-prob, prob) - 1
knn_roc_10 <- evalmod(scores = prob, labels = mh_test$vote96)
autoplot(knn_roc_10)
text(locator(), labels = "KNN ROC, K = 10")
```

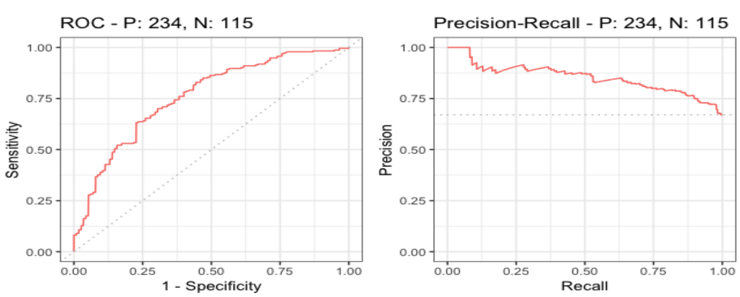
Logit ROC



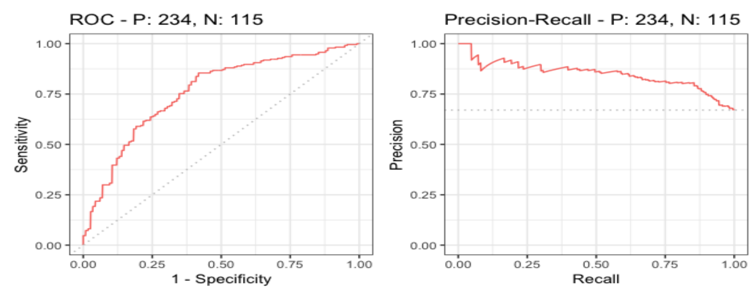
LDA ROC



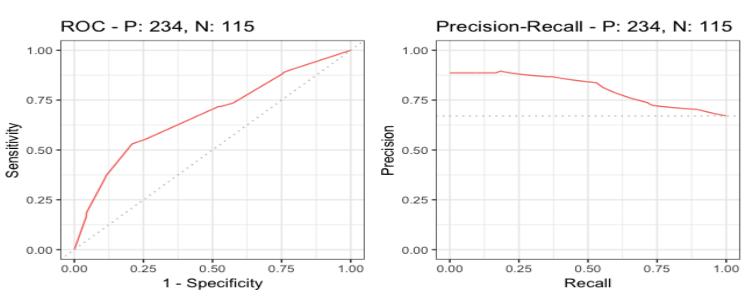
QDA ROC



Naive Bayes ROC



KNN ROC, K = 10



KNN (k = 10)

```
  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.6757711
2         m1          1        PRC 0.8071698
```

```
=== AUCs ===          Logit

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.7816425
2         m1          1        PRC 0.8541451

=== AUCs ===          LDA

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.7790041
2         m1          1        PRC 0.8529546
=== AUCs ===          QDA

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.7587886
2         m1          1        PRC 0.8529507
=== AUCs ===          Naive Bayes

  Model name Dataset ID Curve type       AUC
1         m1          1        ROC 0.7581940
2         m1          1        PRC 0.8492663
```

(d) Our evaluation of best model performance relies on two criteria: the test error rate and the AUC. A lower test error rate indicates higher accuracy, and a higher AUC indicates that the model is better at distinguishing between classes. Based on these two criteria, we notice:

Logit AUC: 0.7816
Logit Test Error: 0.2473

	Logit Error	LDA Error	QDA Error	Naive Bayes Error	KNN (k = 10) Best Error
[1,]	0.2473265	0.2664756	0.2578797	0.252149	0.2893983

These are the lowest among all of the models we observed. Therefore, based on the models we observed, logistic regression performs the best. However, it’s important to note that it’s possible that a low test error doesn’t necessarily entail a high AUC. If these two estimates diverged and there were no single model that had both highest AUC and lowest test error, then we would have to select a model based on our priorities: representativeness versus distinguishing power between classes, respectively.