Homework 05

STAT 430, Fall 2017

Due: Friday, October 13, 11:59 PM

Exercise 1 (Detecting Cancer with KNN)

[7 points] For this exercise we will use data found in `wisc-trn.csv` and `wisc-tst.csv` which contain train and test data respectively. `wisc.csv` is provided but not used. This is a modification of the Breast Cancer Wisconsin (Diagnostic) dataset from the UCI Machine Learning Repository. Only the first 10 feature variables have been provided. (And these are all you should use.)

- UCI Page
- Data Detail

You should consider coercing the response to be a factor variable. Use KNN with all available predictors. For simplicity, do not scale the data. (In practice, scaling would slightly increase performance on this dataset.) Consider $k = 1, 3, 5, 7, \ldots, 51$. Plot train and test error vs k on a single plot.

Use the seed value provided below for this exercise.

Solution:

Note that some code, for plotting and summarizing, is hidden. See the .Rmd file for code.

```
set.seed(314)
```

```
# import data
wisc_trn = read.csv("wisc-trn.csv")
wisc_tst = read.csv("wisc-tst.csv")
```

```
# coerce to factor
wisc_trn$class = as.factor(wisc_trn$class)
wisc_tst$class = as.factor(wisc_tst$class)
```

```
# training data
X_wisc_trn = wisc_trn[, -1]
y_wisc_trn = wisc_trn$class

# testing data
X_wisc_tst = wisc_tst[, -1]
y_wisc_tst = wisc_tst$class
```

```
library(class)
```

```
# error function
calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
```

```
# setup results storage
k_to_try = seq(1, 51, by = 2)
tst_err_k = rep(x = 0, times = length(k_to_try))
trn_err_k = rep(x = 0, times = length(k_to_try))
```

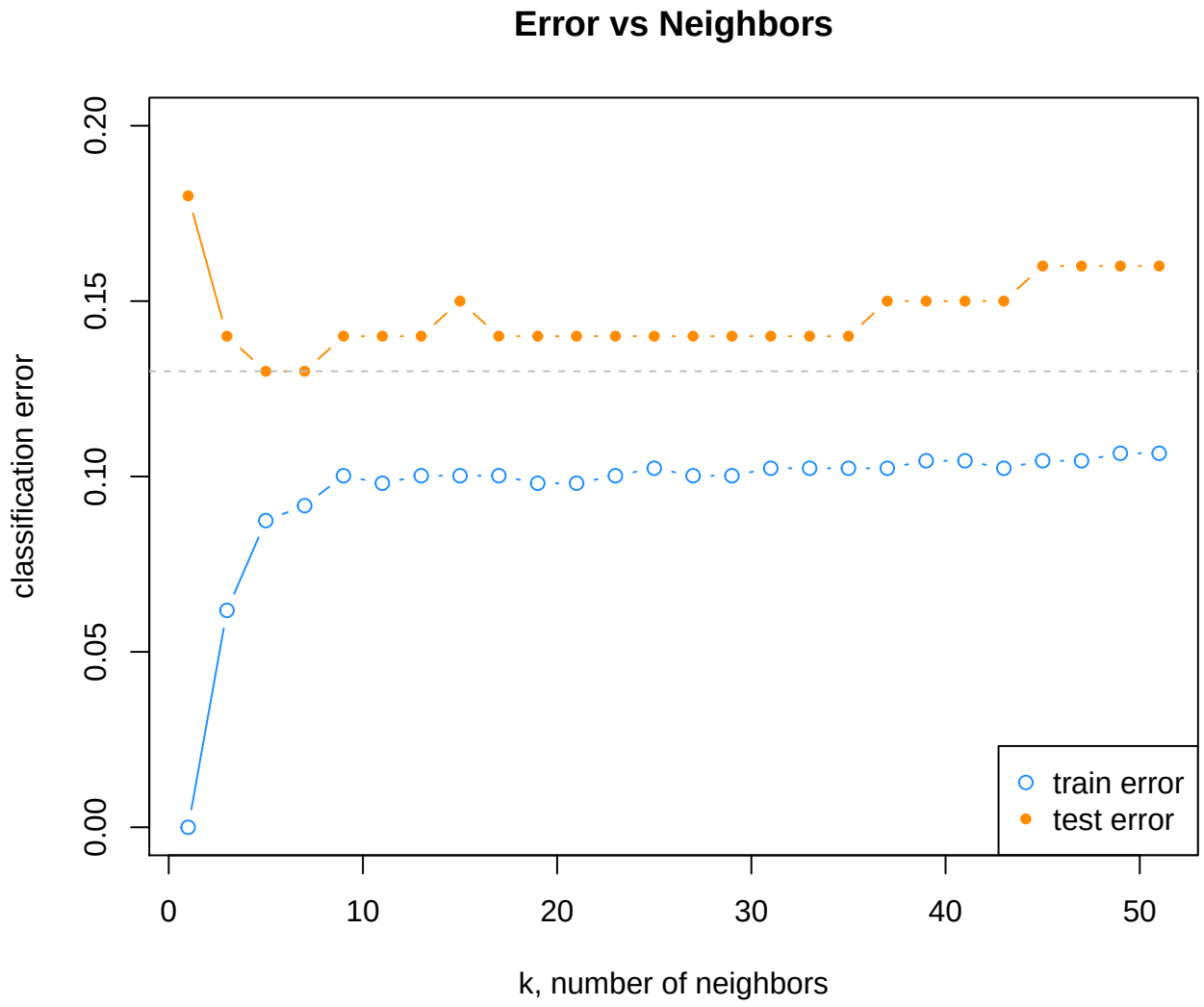
```
# train scaled
for (i in seq_along(k_to_try)) {

  tst_pred = knn(train = X_wisc_trn,
                 test  = X_wisc_tst,
                 cl    = y_wisc_trn,
                 k     = k_to_try[i])

  trn_pred = knn(train = X_wisc_trn,
                 test  = X_wisc_trn,
                 cl    = y_wisc_trn,
                 k     = k_to_try[i])

  tst_err_k[i] = calc_class_err(y_wisc_tst, tst_pred)
  trn_err_k[i] = calc_class_err(y_wisc_trn, trn_pred)

}
```



Exercise 2 (Logistic Regression Decision Boundary)

[**5 points**] Continue with the cancer data from Exercise 1. Now consider an additive logistic regression that considers only two predictors, `radius` and `symmetry`. Plot the test data with `radius` as the x axis, and `symmetry` as the y axis, with the points colored according to their tumor status. Add a line which represents the decision boundary for a classifier using 0.5 as a cutoff for predicted probability.

Solution:

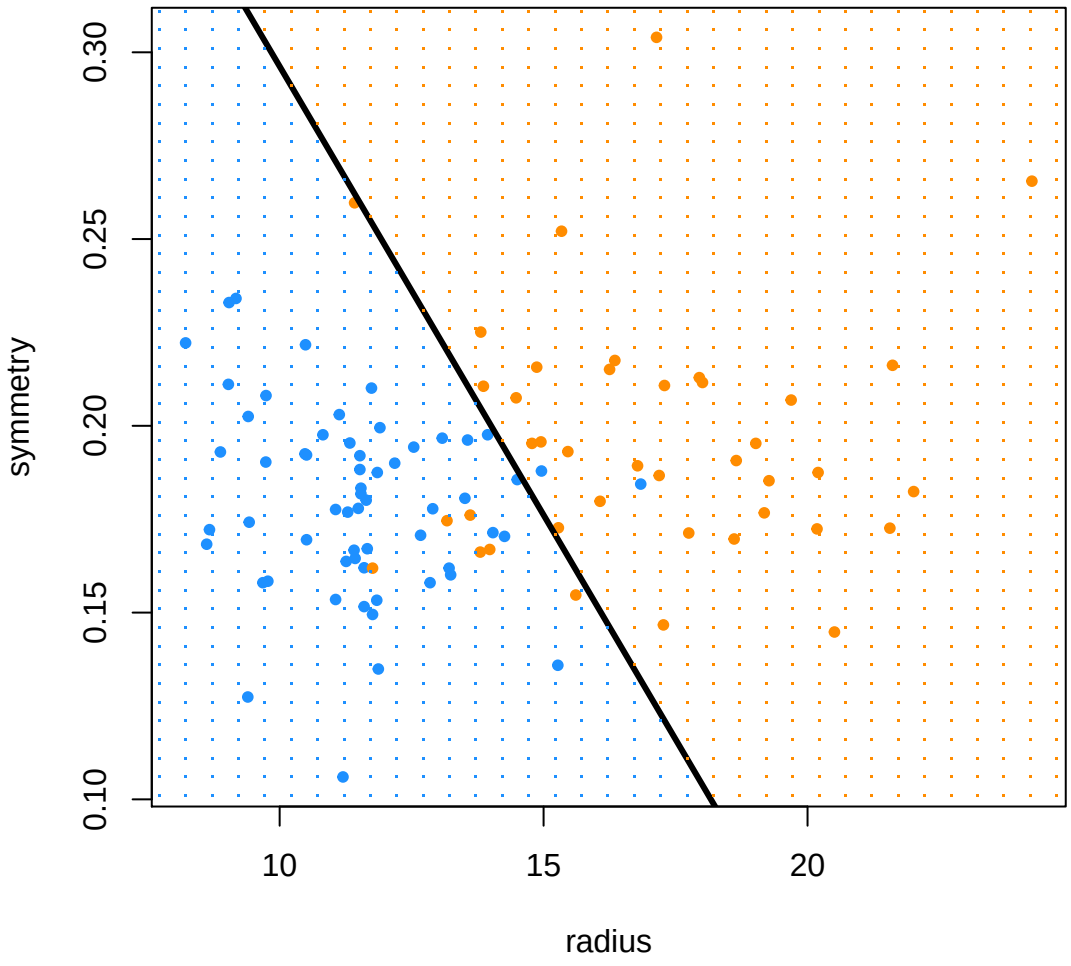
```
wisc_glm = glm(class ~ radius + symmetry, data = wisc_trn, family = "binomial")
```

```
glm_boundary_line = function(glm_fit) {
  intercept = as.numeric(-coef(glm_fit)[1] / coef(glm_fit)[3])
  slope = as.numeric(-coef(glm_fit)[2] / coef(glm_fit)[3])
  c(intercept = intercept, slope = slope)
}

add_glm_boundary = function(glm_fit, line_col = "black") {
  abline(glm_boundary_line(glm_fit), col = line_col, lwd = 3)
}
```

```
a = glm_boundary_line(wisc_glm)[1]
b = glm_boundary_line(wisc_glm)[2]
rad = seq(min(wisc_tst$radius) - 5, max(wisc_tst$radius) + 5, by = 0.5)
sym = seq(min(wisc_tst$symmetry) - 1, max(wisc_tst$symmetry) + 1, by = 0.005)
grid = expand.grid(rad = rad, sym = sym)
bgcol = ifelse(grid$sym > a + b * grid$rad, "darkorange", "dodgerblue")
class_col = ifelse(wisc_tst$class == "M", "darkorange", "dodgerblue")

plot(symmetry ~ radius, data = wisc_tst, col = class_col, pch = 20)
add_glm_boundary(wisc_glm)
points(expand.grid(rad, sym), col = bgcol, pch = ".")
```



Exercise 3 (Sensitivity and Specificity of Cancer Detection)

[**5 points**] Continue with the cancer data from Exercise 1. Again consider an additive logistic regression that considers only two predictors, `radius` and `symmetry`. Report test sensitivity, test specificity, and test accuracy for three classifiers, each using a different cutoff for predicted probability:

- $c = 0.1$
- $c = 0.5$
- $c = 0.9$

Consider `M` to be the "positive" class when calculating sensitivity and specificity. Summarize these results using a single well-formatted table.

Solution:

```
wisc_glm = glm(class ~ radius + symmetry, data = wisc_trn, family = "binomial")
```

```
get_pred = function(mod, data, res = "y", pos = 1, neg = 0, cut = 0.5) {
  probs = predict(mod, newdata = data, type = "response")
  ifelse(probs > cut, pos, neg)
}
```

```r
pred_10 = get_pred(wisc_glm, wisc_tst, res = "class", cut = 0.1, pos = "M", neg = "B")
pred_50 = get_pred(wisc_glm, wisc_tst, res = "class", cut = 0.5, pos = "M", neg = "B")
pred_90 = get_pred(wisc_glm, wisc_tst, res = "class", cut = 0.9, pos = "M", neg = "B")

tab_10 = table(predicted = pred_10, actual = wisc_tst$class)
tab_50 = table(predicted = pred_50, actual = wisc_tst$class)
tab_90 = table(predicted = pred_90, actual = wisc_tst$class)

con_mat_10 = caret::confusionMatrix(tab_10, positive = "M")
con_mat_50 = caret::confusionMatrix(tab_50, positive = "M")
con_mat_90 = caret::confusionMatrix(tab_90, positive = "M")

metrics = rbind(

  c(con_mat_10$overall["Accuracy"],
    con_mat_10$byClass["Sensitivity"],
    con_mat_10$byClass["Specificity"]),

  c(con_mat_50$overall["Accuracy"],
    con_mat_50$byClass["Sensitivity"],
    con_mat_50$byClass["Specificity"]),

  c(con_mat_90$overall["Accuracy"],
    con_mat_90$byClass["Sensitivity"],
    con_mat_90$byClass["Specificity"])

)

rownames(metrics) = c("c = 0.10", "c = 0.50", "c = 0.90")
knitr::kable(metrics)
```

	Accuracy	Sensitivity	Specificity
c = 0.10	0.86	0.950	0.8000000
c = 0.50	0.91	0.825	0.9666667
c = 0.90	0.82	0.575	0.9833333

Exercise 4 (Comparing Classifiers)

```r
# load libraries
library(MASS)

# setup parameters
num_obs = 1000

# means
mu_1 = c(12, 8.5)
mu_2 = c(22, 10)
mu_3 = c(12, 15)
```

```r
mu_4 = c(12, 20)

# sigmas
sigma_1 = matrix(c(10, -4, -4, 8), 2, 2)
sigma_2 = matrix(c(5, -3, -3, 5), 2, 2)
sigma_3 = matrix(c(8, 3, 3, 8), 2, 2)
sigma_4 = matrix(c(8, 6, 6, 8), 2, 2)

# control randomization
set.seed(42)

# make train data
hw05_trn = data.frame(

  # create response
  as.factor(c(rep("A", num_obs / 2), rep("B", num_obs),
              rep("C", num_obs * 2), rep("D", num_obs))),

  # create predictors
  rbind(
    mvrnorm(n = num_obs / 2, mu = mu_1, Sigma = sigma_1),
    mvrnorm(n = num_obs, mu = mu_2, Sigma = sigma_2),
    mvrnorm(n = num_obs * 2, mu = mu_3, Sigma = sigma_3),
    mvrnorm(n = num_obs, mu = mu_4, Sigma = sigma_4)
  )

)
# label variables
colnames(hw05_trn) = c("y", "x1", "x2")


# make test data
hw05_tst = data.frame(

  # create response
  as.factor(c(rep("A", num_obs), rep("B", num_obs),
              rep("C", num_obs), rep("D", num_obs))),

  # create predictors
  rbind(
    mvrnorm(n = num_obs, mu = mu_1, Sigma = sigma_1),
    mvrnorm(n = num_obs, mu = mu_2, Sigma = sigma_2),
    mvrnorm(n = num_obs, mu = mu_3, Sigma = sigma_3),
    mvrnorm(n = num_obs, mu = mu_4, Sigma = sigma_4)
  )

)
# label variables
colnames(hw05_tst) = c("y", "x1", "x2")

# write to files
readr::write_csv(hw05_trn, "hw05-trn.csv")
readr::write_csv(hw05_tst, "hw05-tst.csv")
```

```
# clear workspace
rm(list = ls())
```

[**7 points**] Use the data found in hw05-trn.csv and hw05-tst.csv which contain train and test data respectively. Use `y` as the response. Coerce `y` to be a factor after importing the data if it is not already.

Create pairs plot with ellipses for the training data, then train the following models using both available predictors:

- Additive Logistic Regression
- LDA (with Priors estimated from data)
- LDA with Flat Prior
- QDA (with Priors estimated from data)
- QDA with Flat Prior
- Naive Bayes (with Priors estimated from data)

Calculate test and train error rates for each model. Summarize these results using a single well-formatted table.

Solution:

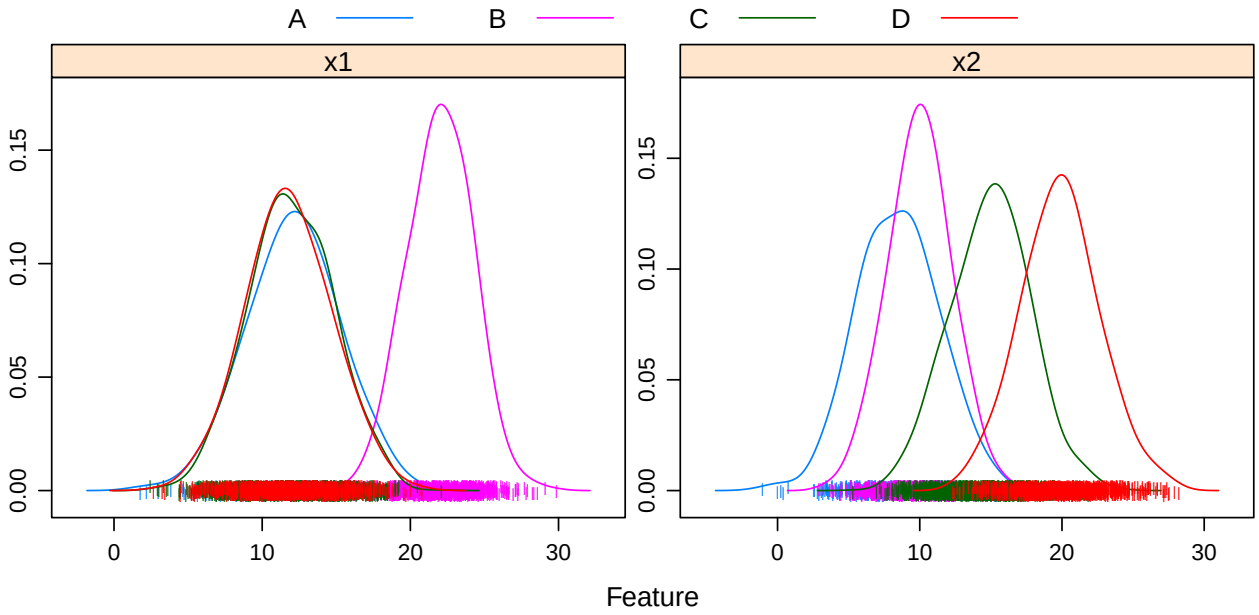
```
# read data
hw05_trn = readr::read_csv("hw05-trn.csv")
hw05_tst = readr::read_csv("hw05-tst.csv")

# coerce characters to factors
hw05_trn$y = as.factor(hw05_trn$y)
hw05_tst$y = as.factor(hw05_tst$y)
```

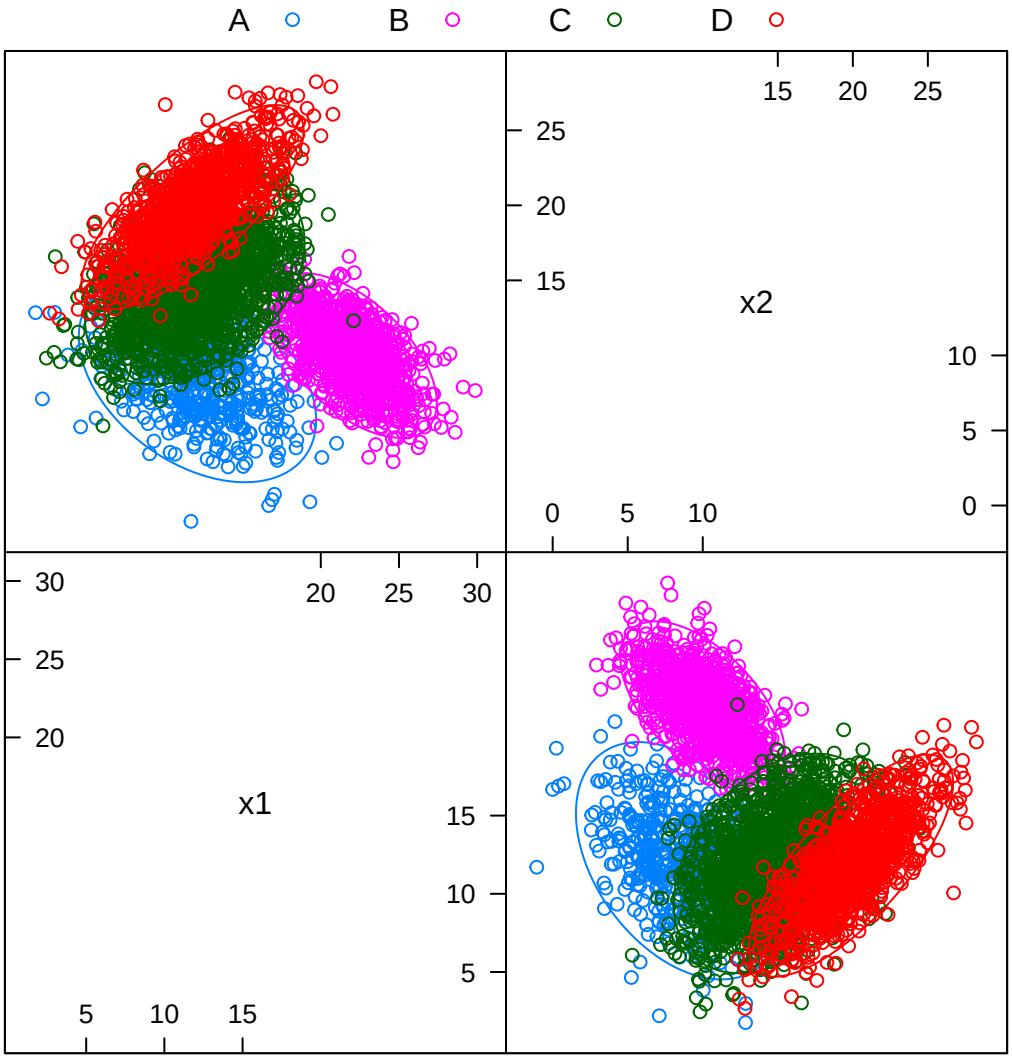
Note: when using the older `read.csv()` strings are *automatically* imported as factors by deafult. This would seem useful here, but a terrible idea in general. It is better to import as a character, then later explicitly coerce to a factor if desired. For this reason, `read_csv()` does not even provide an option to import characters as a factor. (At least not one this instructor is aware of.)

```
# load packages
library(MASS)
library(e1071)
library(caret)
library(nnet)
library(ellipse)
```

```
caret::featurePlot(x = hw05_trn[, 2:3],
                   y = hw05_trn$y,
                   plot = "density",
                   scales = list(x = list(relation = "free"),
                                 y = list(relation = "free")),
                   adjust = 1.5,
                   pch = "|",
                   layout = c(2, 1),
                   auto.key = list(columns = 4))
```




```
featurePlot(x = hw05_trn[, 2:3],
            y = hw05_trn$y,
            plot = "ellipse",
            auto.key = list(columns = 4))
```



Scatter Plot Matrix

```
# error function
calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
```

```
# classifiers to be used
hw05_classifiers = c("Logistic", "LDA", "LDA, Flat Prior",
                     "QDA", "QDA, Flat Prior", "Naive Bayes")

# define flat prior
flat = c(1, 1, 1, 1) / 4

# calcualte train errors
hw05_trn_err = c(
  calc_class_err(hw05_trn$y, predict(multinom(y ~ ., hw05_trn, trace = FALSE), hw05_trn)),
  calc_class_err(hw05_trn$y, predict(lda(y ~ ., hw05_trn), hw05_trn)$class),
  calc_class_err(hw05_trn$y, predict(lda(y ~ ., hw05_trn, prior = flat), hw05_trn)$class),
  calc_class_err(hw05_trn$y, predict(qda(y ~ ., hw05_trn), hw05_trn)$class),
  calc_class_err(hw05_trn$y, predict(qda(y ~ ., hw05_trn, prior = flat), hw05_trn)$class),
```

```
  calc_class_err(hw05_trn$y, predict(naiveBayes(y ~ ., hw05_trn), hw05_trn))
)

# calcualte test errors
hw05_tst_err = c(
  calc_class_err(hw05_tst$y, predict(multinom(y ~ ., hw05_trn, trace = FALSE), hw05_tst)),
  calc_class_err(hw05_tst$y, predict(lda(y ~ ., hw05_trn), hw05_tst)$class),
  calc_class_err(hw05_tst$y, predict(lda(y ~ ., hw05_trn, prior = flat), hw05_tst)$class),
  calc_class_err(hw05_tst$y, predict(qda(y ~ ., hw05_trn), hw05_tst)$class),
  calc_class_err(hw05_tst$y, predict(qda(y ~ ., hw05_trn, prior = flat), hw05_tst)$class),
  calc_class_err(hw05_tst$y, predict(naiveBayes(y ~ ., data = hw05_trn), hw05_tst))
)

# store results in data frame
hw05_results = data.frame(
  hw05_classifiers,
  hw05_trn_err,
  hw05_tst_err
)

# create column titles
colnames(hw05_results) = c("Method", "Train Error", "Test Error")
```

```
# display data frame as table
knitr::kable(hw05_results)
```

Method	Train Error	Test Error
Logistic	0.1482222	0.17425
LDA	0.1620000	0.19825
LDA, Flat Prior	0.1906667	0.16875
QDA	0.1477778	0.16925
QDA, Flat Prior	0.1791111	0.14000
Naive Bayes	0.1733333	0.20000

Exercise 5 (Concept Checks)

[**1 point each**] Answer the following questions based on your results from the three exercises.

(a) Which k performs best in Exercise 1?

Solution: Based on the plot, we could choose $k = 7$.

```
k_to_try[max(which(tst_err_k == min(tst_err_k)))]
```

```
## [1] 7
```

(b) In Exercise 4, which model performs best?

Solution: We see that the QDA with a Flat Prior performs the best.

(c) In Exercise 4, why does Naive Bayes perform poorly?

Solution: The plot offers intuition for QDA > LDA > NB. NB performs the worst because there is clearly significant correlation between `x1` and `x2` in all classes. (See the data generation code.)

(d) In Exercise 4, which performs better, LDA or QDA? Why?

Solution: Again, the plot offeres intuition. Between LDA and QDA it is clear that QDA is better as the Σ_k appear to be very different for different classes. (See the data generation code.)

(e) In Exercise 4, which prior performs better? Estimating from data, or using a flat prior? Why?

Solution: The flat prior. The fact that the Flat Prior works best doesn't have any intuition here, since there is no context. It just so happens that the proportion of classes in the test data is uniform. (See the data generation code.)

```
# class proportions in train data
table(hw05_trn$y) /  length(hw05_trn$y)
```

```
##
##         A         B         C         D
## 0.1111111 0.2222222 0.4444444 0.2222222
```

```
# class proportions in test data
table(hw05_tst$y) /  length(hw05_tst$y)
```

```
##
##    A    B    C    D
## 0.25 0.25 0.25 0.25
```

(f) In Exercise 4, of the four classes, which is the easiest to classify?

Solution: Class `B`. From the confusion matrix, we see that QDA with Flat Prior is predicting best inside of class `B`. It has by far the fewest results off the diagonal. This is unsurprising as we could see from the pairs plot that the `B` class had the least overlap with the other classes. This is mostly due to its values of `x1`.

```
# confusion matrix
table(predicted = predict(qda(y ~ ., data = hw05_trn,
                              prior = c(1, 1, 1, 1) / 4),
                          hw05_tst)$class,
      actual    = hw05_tst$y)
```

```
##          actual
## predicted   A   B   C   D
##         A 876   6 126   6
##         B   5 982  11   0
##         C 104  12 701 113
##         D  15   0 162 881
```

(g) [**Not Graded**] In Exercise 3, which classifier would be the best to use in practice?

Solution: It depends. Do you want more false negatives (tellings people they don't have cancer when they actually do) or more false positives (telling people they have cancer when they actually do not)? A high sensitivity will give use few false negatives. A high specificity will give us few false positives. Based on our results, we see that there is a trade-off, and we can't have it both ways.