Problem 1.

Take the following training/test split: 13000 training and 6020 test. Perform all necessary tests, plots, and Monte Carlo simulations to determine your final choice of classifier. You can use the built-in function of your favorite program. Can you beat the 86.6% mean accuracy on test data (based on 100 runs)?

Solution

```r
#########
# Setup #
#########

library(foreach)
library(doParallel)

data <- read.csv("magic04.data", header=F, sep=",")
train.size <- 13000 # Given by homework specification

#######
# KNN #
#######

# Try a bunch of different K-values,
err <- foreach (K=1:50, .combine = c) %do% {
    cl <- makeCluster(4)
    registerDoParallel(cl)

    # Run KNN 100 times for each K value.
    # Each run is independent, so we can speed things up a little
    # bit by running it in parallel.
    k.err <- foreach (i=1:100, .combine = c) %dopar% {
        # Need to load the library for knn on each thread.
        library(class)

        data <- data[sample(nrow(data)),] # Randomize the data set

        train <- data[1:train.size, 1:10]
        test <- data[(train.size+1):nrow(data), 1:10]
        train.cl <- factor(data[1:train.size, 11])
        test.cl <- factor(data[(train.size+1):nrow(data), 11]);

        predict.cl <- knn(train, test, train.cl, k=K)
        sum(test.cl != predict.cl) / nrow(test)
```

```
    }

    stopCluster(cl)
    mean(k.err)
}

# This was our best performing k value.
k <- which.min(err)
min.err <- min(err)
acc <- 1.0 - min.err

# About 80.975%
print(paste("KNN - Accuracy: ", acc))

#######
# LDA #
#######

# cl <- makeCluster(4)
# registerDoParallel(cl)
#
# err <- foreach (i=1:100, .combine = c) %dopar% {
#     library(MASS)
#
#     data <- data[sample(nrow(data)),] # Randomize the data set
#
#     train <- data[1:train.size, 1:10]
#     test <- data[(train.size+1):nrow(data), 1:10]
#     train.cl <- factor(data[1:train.size, 11])
#     test.cl <- factor(data[(train.size+1):nrow(data), 11]);
#
#     model <- lda(x = train, grouping = train.cl)
#     predict.cl <- predict(model, test)£class
#     sum(test.cl != predict.cl) / nrow(test)
# }
#
# stopCluster(cl)
# acc <- 1.0 - mean(err)
#
# # About 78.429%
# print(paste("LDA - Accuracy: ", acc))

#######
# QDA #
#######
```

```
# cl <- makeCluster(4)
# registerDoParallel(cl)
#
# err <- foreach (i=1:100, .combine = c) %dopar% {
#     library(MASS)
#
#     data <- data[sample(nrow(data)),] # Randomize the data set
#
#     train <- data[1:train.size, 1:10]
#     test <- data[(train.size+1):nrow(data), 1:10]
#     train.cl <- factor(data[1:train.size, 11])
#     test.cl <- factor(data[(train.size+1):nrow(data), 11]);
#
#     model <- qda(x = train, grouping = train.cl)
#     predict.cl <- predict(model, test)£class
#     sum(test.cl != predict.cl) / nrow(test)
# }
#
# stopCluster(cl)
# acc <- 1.0 - mean(err)
#
# # About 78.4276%
# print(paste("QDA - Accuracy: ", acc))

########################
# Naive Bayes (Normal) #
########################

# cl <- makeCluster(4)
# registerDoParallel(cl)
#
# err <- foreach (i=1:100, .combine = c) %dopar% {
#     library(klaR)
#     library(caret)
#
#     data <- data[sample(nrow(data)),] # Randomize the data set
#
#     train <- data[1:train.size, 1:10]
#     test <- data[(train.size+1):nrow(data), 1:10]
#     train.cl <- factor(data[1:train.size, 11])
#     test.cl <- factor(data[(train.size+1):nrow(data), 11]);
#
#     model <- NaiveBayes(x = train, grouping = train.cl, usekernel=FALSE)
#     predict.cl <- predict(model, test)£class
```

```
#     sum(test.cl != predict.cl) / nrow(test)
# }
#
# stopCluster(cl)
# acc <- 1.0 - mean(err)
#
# # About 72.6714%
# print(paste("Naive Bayes (Normal) - Accuracy: ", acc))

#######################
# Naive Bayes (Kernel) #
#######################

# cl <- makeCluster(4)
# registerDoParallel(cl)
#
# err <- foreach (i=1:100, .combine = c) %dopar% {
#     library(klaR)
#     library(caret)
#
#     data <- data[sample(nrow(data)),] # Randomize the data set
#
#     train <- data[1:train.size, 1:10]
#     test <- data[(train.size+1):nrow(data), 1:10]
#     train.cl <- factor(data[1:train.size, 11])
#     test.cl <- factor(data[(train.size+1):nrow(data), 11]);
#
#     model <- NaiveBayes(x = train, grouping = train.cl, usekernel=TRUE)
#     predict.cl <- predict(model, test)£class
#     sum(test.cl != predict.cl) / nrow(test)
# }
#
# stopCluster(cl)
# acc <- 1.0 - mean(err)
#
# # About 76.2375%
# print(paste("Naive Bayes (Kernel) - Accuracy: ", acc))
```

Problem 2.

(a) Classify the test point (0, 1) using QDA and calculate the posterior class probabilities. Do the calculations by hand.

(b) Classify the test point (0, 1) using naive Bayes assuming normality and calculate the posterior class probabilities. Do the calculations by hand.

(c) Verify your results for both classifiers using Matlab, R, etc. You can use the built-in functions.

Solution

Part (a)

First we need to calculate the class means $\hat{\mu}_0$ and $\hat{\mu}_1$

$$
\begin{aligned}
\hat{\mu}_0 &= \frac{1}{3}([0.6585, 0.2444] + [2.2460, 0.5281] + [-2.7665, -3.8303]) \\
&= \frac{1}{3}[0.138, -3.8303] \\
&= [0.046, -1.019267] \\
\hat{\mu}_1 &= \frac{1}{3}([-1.2565, 3.4912] + [-0.7973, 1.2288] + [1.1170, 2.2637]) \\
&= \frac{1}{3}[-0.9368, 6.9837] \\
&= [-0.3122667, 2.3279000]
\end{aligned}
$$

Next calculate the covariance matrix for each class.

$$
\begin{aligned}
\hat{\Sigma}_0 = &\frac{1}{2}((([0.6585, 0.2444] - \hat{\mu}_0)([0.6585, 0.2444] - \hat{\mu}_0)^\intercal \\
&+ ([2.2460, 0.5281] - \hat{\mu}_0)([2.2460, 0.5281] - \hat{\mu}_0)^\intercal \\
&+ ([-2.7665, -3.8303] - \hat{\mu}_0)([-2.7665, -3.8303] - \hat{\mu}_0)^\intercal \\
&= \frac{1}{2}(\begin{bmatrix} 0.33516 & 0.77400 \\ 0.77400 & 1.59685 \end{bmatrix} + \begin{bmatrix} 4.8400 & 3.4042 \\ 3.4042 & 2.394 \end{bmatrix} + \begin{bmatrix} 7.9102 & 7.9060 \\ 7.9060 & 7.9019 \end{bmatrix}) \\
&= \begin{bmatrix} 6.5627 & 6.0421 \\ 6.0421 & 5.9466 \end{bmatrix}
\end{aligned}
$$

$$\begin{aligned}
\hat{\Sigma}_1 &= \frac{1}{2}(([-1.2565, 3.4912] - \hat{\mu}_1)([-1.2565, 3.4912] - \hat{\mu}_1)^\intercal \\
&+ ([-0.7973, 1.2288] - \hat{\mu}_1)([-0.7973, 1.2288] - \hat{\mu}_1)^\intercal \\
&+ ([1.1170, 2.2637] - \hat{\mu}_1)([1.1170, 2.2637] - \hat{\mu}_1)^\intercal \\
&= \frac{1}{2}(\begin{bmatrix} 0.89158 & -1.09843 \\ -1.09843 & 1.35327 \end{bmatrix} + \begin{bmatrix} 0.23526 & 0.53310 \\ 0.53310 & 1.20802 \end{bmatrix} + \begin{bmatrix} 2.0426033 & -0.0917589 \\ -0.0917589 & 2.0426033 \end{bmatrix}) \\
&= \begin{bmatrix} 1.58482 & -0.32854 \\ -0.32854 & 1.28270 \end{bmatrix}
\end{aligned}$$

We would like to maximize $\hat{P}[Y = k]\,\hat{f}(X = x|Y = k)$ over $k \in \{0, 1\}$ for our input x=(0,1).

$$\arg\max_{k \in \{0,1\}} \hat{P}[Y = k]\left(\frac{1}{(2\pi)^{\frac{d}{2}}|\hat{\Sigma}_k|^{\frac{1}{2}}}\right) \exp(\frac{1}{2}(x - \hat{\mu}_k)^\intercal \hat{\Sigma}_k^{-1}(x - \hat{\mu}_k))$$

case: k = 0

$$\begin{aligned}
\hat{P}[Y = 0] = \frac{3}{6} = \frac{1}{2} \\
|\hat{\Sigma}_0| = 2.5180 \Rightarrow |\hat{\Sigma}_0|^{\frac{1}{2}} = 1.5868 \\
\left(\frac{1}{(2\pi)^{\frac{2}{2}}|\hat{\Sigma}_0|^{\frac{1}{2}}}\right) = (\frac{1}{9.9702}) = 0.10030 \\
\exp(\frac{1}{2}(x - \hat{\mu}_0)^\intercal \hat{\Sigma}_0^{-1}(x - \hat{\mu}_0)) = \\
\exp(-\frac{1}{2} * 11.078) = \exp(-5.5389) = 0.0039308 \\
\hat{P}[Y = 0]\,\hat{f}(X = x|Y = 0) = .00019713
\end{aligned}$$

case: k = 1

$$\begin{aligned}
\hat{P}[Y = 1] = \frac{3}{6} = \frac{1}{2} \\
|\hat{\Sigma}_1| = 1.9249 \Rightarrow |\hat{\Sigma}_1|^{\frac{1}{2}} = 1.3874 \\
\left(\frac{1}{(2\pi)^{\frac{2}{2}}|\hat{\Sigma}_1|^{\frac{1}{2}}}\right) = \frac{1}{8.7174} = 0.11471 \\
\exp(-\frac{1}{2}(x - \hat{\mu}_1)^\intercal \hat{\Sigma}_1^{-1}(x - \hat{\mu}_1)) = \\
\exp(-\frac{1}{2} * 1.3752) = \exp(-0.6876) = 0.50278 \\
\hat{P}[Y = 1]\,\hat{f}(X = x|Y = 1) = 0.028838
\end{aligned}$$

Noting that $0.00019713 < 0.028838$, k=1 maximizes our function, thus we predict a class label of 1 for (0,1). The posterior class probability is given by the following function.

$$
\begin{aligned}
P[Y=k|X=x] &= \frac{P[Y=k]f_{1\ldots d}(X=x|Y=k)}{\sum_{i=0}^{k} P[Y=i]f_{Y\ldots d}(X=x|Y=i)} \\
&= \frac{0.028838}{0.028838+0.00019713} \\
&= 0.99321
\end{aligned}
$$

Thus we find that we have a posterior class probability of 99.321% for class k=1.

Part (b)

The naive Bayes classification uses the same class norms $\hat{\mu}_0$ and $\hat{\mu}_1$.
However we will need to compute $\hat{\sigma}^2$ values for each feature of each class.

case: k = 0
First compute the variance of each feature.

$$
\begin{aligned}
\hat{\sigma}_0^2 &= \frac{1}{3} * \sum_{j \in C_0} (x_i - \hat{\mu}_i)^2 \\
&= \frac{1}{3}(((0.6585, 0.2444) - \hat{\mu}_0)^2 + ((2.2460, 0.5281) - \hat{\mu}_0)^2 + ((-2.7665, -3.8303) - \hat{\mu}_0)^2) \\
&= \frac{1}{3}(13.125, 11.893) \\
&= (4.3751, 3.9644)
\end{aligned}
$$

Now we can compute $f_i((0,1)|Y=0)$ for use in our classifier.

$$
\begin{aligned}
f((0,1)|Y=0) &= \frac{1}{\sqrt{2\hat{\sigma}_0^2\pi}} e^{-\frac{((0,1)-\hat{\mu}_0)^2}{2\hat{\sigma}_0^2}} \\
&= \frac{1}{(5.2431, 4.9909)}(0.99976, 0.59794) \\
&= (0.19068, 0.11981)
\end{aligned}
$$

We would like to maximize the following equation over all k $\hat{P}[Y=0]\prod_{i=1}^{d} f_i(X_i|Y=k)$.

$$
\begin{aligned}
\hat{P}[Y=0] &= \frac{3}{6} \\
f_0(X_0|Y=0) &= 0.19068 \\
f_1(X_1|Y=0) &= 0.11981 \\
\frac{1}{2} * 0.19068 * 0.11981 &= 0.011423
\end{aligned}
$$

case: k = 1
First compute the variance of each feature.

$$\begin{aligned}
\hat{\sigma}_1^2 &= \frac{1}{3} * \sum_{j \in C_1} (x_i - \hat{\mu}_i)^2 \\
&= \frac{1}{3}(((-1.2565, 3.4912) - \hat{\mu}_1)^2 + ((-0.7973, 1.2288) - \hat{\mu}_1)^2 + (1.1170, 2.2637) - \hat{\mu}_1)^2) \\
&= \frac{1}{3}(3.1696, 2.5654) \\
&= (1.05655, 0.85514)
\end{aligned}$$

Now we can compute $f_i((0,1)|Y=1)$ for use in our classifier.

$$\begin{aligned}
f((0,1)|Y=1) &= \frac{1}{\sqrt{2\hat{\sigma}_1^2 \pi}} e^{-\frac{((0,1)-\hat{\mu}_1)^2}{2\hat{\sigma}_1^2}} \\
&= \frac{1}{(2.5765, 2.3180)}(0.95490, 0.35664) \\
&= (0.37062, 0.15386)
\end{aligned}$$

We would like to maximize the following equation over all k $\hat{P}[Y=1] \prod_{i=1}^{d} f_i(X_i|Y=k)$.

$$\begin{aligned}
\hat{P}[Y=1] &= \frac{3}{6} \\
f_0(X_0|Y=0) &= 0.37062 \\
f_1(X_1|Y=0) &= 0.15386 \\
\frac{1}{2} * 0.37062 * 0.15386 &= 0.028512
\end{aligned}$$

Noting that $0.011423 < 0.028512$, k=1 maximizes our function, thus we predict a class label of 1 for (0,1). The posterior class probability is given by the following function.

$$\begin{aligned}
P[Y=k|X=x] &= \frac{P[Y=k] f_{1\ldots d}(X=x|Y=k)}{\sum_{i=0}^{k} P[Y=i] f_{Y\ldots d}(X=x|Y=i)} \\
&= \frac{0.028512}{0.028512 + 0.011423} \\
&= 0.71396
\end{aligned}$$

Thus we find that we have a posterior class probability of 71.396% for class k=1.

Part (c)

I have included the results of the R script that follows as comments directly below the corresponding print statements. The result of qda(...) in R produces the exact same posterior class probability as I have computed above. Currently my results for NaiveBayes(...) are off by about 4%, so I must have an error in my work somewhere.

```r
#########
# Setup #
#########

library(MASS)
library(klaR)

data <- data.frame(
    c(0.6585, 2.2460, -2.7665, -1.2565, -0.7973, 1.1170),
    c(0.2444, 0.5281, -3.8303, 3.4912, 1.2288, 2.2637),
    c(0, 0, 0, 1, 1, 1)
)

test <- data.frame(c(0), c(1))

names(data) <- c("F1", "F2", "CLASS")
names(test) <- c("F1", "F2")

train.size <- nrow(data)
train <- data[1:train.size, 1:2]
train.cl <- factor(data[1:train.size, 3])

#######
# QDA #
#######

model <- qda(x = train, grouping = train.cl)
predict <- predict(model, test)
print(predict)
print(paste("QDA: ", predict$class))

# Posterior Class Probabilities
# 0: 0.0067895
# 1: 0.9932105

###############
# Naive Bayes #
###############
```

```

model <- NaiveBayes(x = train, grouping = train.cl, usekernel=FALSE)
predict <- predict(model, test)
print(predict)
print(paste("Naive Bayes: ", predict$class))

# Posterior Class Probabilities
# 0: 0.2493135
# 1: 0.7506865
```