Stat 432 Homework 9

Sharvi Tomar (stomar2)

Assigned: Oct 25, 2021; Due: 11:59 PM CT, Nov 2, 2021

Contents

Question 1: LDA

Let's start with estimating some components in the LDA. First, by the lecture notes, we know that and LDA is to compare the log of densities and the prior probabilities, i.e., for each target point x_0, we want to find the class label k that has the largest value of

$$x_0^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k)$$

Let's use the `SAheart` data from the `ElemStatLearn` package to perform this calculation. In this data, there are two classes, defined by the `chd` (chronic heart disease) variable. And there are 9 variables. We will treat them all as numerical variables, hence the following X and y are used:

```
library(ElemStatLearn)
X = data.matrix(SAheart[, -10])
y = SAheart$chd
```

Hence, the problem is essentially estimating the quantities:

- [5 Points] Prior probabilities π_k

```
# Prior probabilities
pi_0 = length(y[y == "0"]) / length(y)
pi_1 = length(y[y == "1"]) / length(y)

pi_0
```

```
## [1] 0.6536797
```

```
pi_1
```

```
## [1] 0.3463203
```

The prior probabilities are 0.6536797 (y=0 or class 0) and 0.3463203 (y=1 or class 1).

- [10 Points] Mean vectors (centroid) for each class: μ_k

```
# Forming a data frame from X and y
SAheart_final = data.frame(X, y)

# Subsetting the dataframe to have observations belonging to class=0
class_0 = SAheart_final[(SAheart_final$y == "0"),]
# Mean vector for class-0
m0 = apply(class_0[, 1:9], 2, mean)

# Subsetting the dataframe to have observations belonging to class=1
class_1 = SAheart_final[(SAheart_final$y == "1"),]
# Mean vector for class-1
m1 = apply(class_1[, 1:9], 2, mean)
```

```
# Mean vector for class-0
m0
```

```
##        sbp    tobacco        ldl  adiposity    famhist      typea    obesity
## 135.460265   2.634735   4.344238  23.969106   1.317881  52.367550  25.737450
##    alcohol        age
##  15.931358  38.854305
```

The above is the mean vector of class=0.

```
# Mean vector for class-1
m1
```

```
##        sbp    tobacco        ldl  adiposity    famhist      typea    obesity
## 143.737500   5.524875   5.487938  28.120250   1.600000  54.493750  26.622937
##    alcohol        age
##  19.145250  50.293750
```

The above is the mean vector of class=1.

- [20 Points] Pooled covariance matrix Σ

```
# Centering the data belonging to class=0
class_0_centred = data.frame(matrix(NA, nrow = 302, ncol = 9))
for (i in 1:9) {
  class_0_centred[, i] = class_0[, i] - m0[i]
}

# Centering the data belonging to class=1
class_1_centred = data.frame(matrix(NA, nrow = 160, ncol = 9))
for (i in 1:9) {
  class_1_centred[, i] = class_1[, i] - m1[i]
}
```

```r
# Calculating the covariance matrix for variates in class=0
s_0_matrix = t(class_0_centred) %*%  as.matrix(class_0_centred)
# Calculating the covariance matrix for variates in class=1
s_1_matrix = t(class_1_centred) %*%  as.matrix(class_1_centred)

# Calculating pooled covariance matrix
pooled_cov_matrix = (s_0_matrix + s_1_matrix) / (nrow(X) - 2)

pooled_cov_matrix
```

```
##             X1         X2          X3         X4         X5           X6
## X1 405.4347786 14.5851882  4.58124834 49.1643582 0.33698819 -15.58772153
## X2  14.5851882 19.2425564  0.76320271  7.5381093 0.01582050  -2.05730592
## X3   4.5812483  0.7632027  4.00058129  6.0326970 0.09185024   0.34459233
## X4  49.1643582  7.5381093  6.03269701 56.7528931 0.43280832  -5.30956633
## X5   0.3369882  0.0158205  0.09185024  0.4328083 0.22583357   0.08112007
## X6 -15.5877215 -2.0573059  0.34459233 -5.3095663 0.08112007  95.56564291
## X7  18.9388430  1.8334506  2.66005159 22.7078852 0.18402776   2.64006728
## X8  64.4003517 20.5170451 -2.53290675 16.1190253 0.76846885   7.96003164
## X9  95.1338895 22.7654113  6.47892049 60.5086715 0.99736251 -20.27833492
##            X7          X8          X9
## X1 18.9388430  64.4003517  95.1338895
## X2  1.8334506  20.5170451  22.7654113
## X3  2.6600516  -2.5329068   6.4789205
## X4 22.7078852  16.1190253  60.5086715
## X5  0.1840278   0.7684688   0.9973625
## X6  2.6400673   7.9600316 -20.2783349
## X7 17.6154236   4.6893563  15.6969995
## X8  4.6893563 598.2766133  27.8859413
## X9 15.6969995  27.8859413 184.1321373
```

Above is the pooled covariance matrix.

Based on this data, calculate the three components of LDA for each class.

- [20 Points] After calculating these components, use your estimated values to predict the label of each observation in the training data. So this will be the in-sample fitted labels. Provide the confusion table of your results. Please be aware that some of these calculations are based on matrices, hence you must match the dimensions (construct your objects) properly, otherwise, error would occur.

```r
# Initializing a vector to store predicted labels
pred_class = rep(0, nrow(X))

for (i in 1:nrow(X)) {
  ## Dimensions:    1*9               9 * 9                          9*1
  k_0 = -0.5 * (t(X[i, ] - m0) %*% solve(pooled_cov_matrix) %*% as.matrix(X[i, ] - m0)) + log(pi_0)
  k_1 = -0.5 * (t(X[i, ] - m1) %*% solve(pooled_cov_matrix) %*% as.matrix(X[i, ] - m1)) + log(pi_1)

  ## Assigning classes
  if (k_0 > k_1) {
    pred_class[i] = 0
  }
  else{
```

```
    pred_class[i] = 1
  }
}
```

```
# Confusion table of results
table(y, pred_class)
```

```
##    pred_class
## y     0   1
##   0 258  44
##   1  73  87
```

The above is the confusion table of predicted in-sample fitted labels using LDA calculated by estimating its components.

- [5 Points] Perform the same LDA analysis using the built in `lda` function and provide the confusion table. Are these results match?

```
# Performing LDA analysis using built-in `lda` function
library(MASS)
dig.lda = lda(X, y)
Y.pred = predict(dig.lda, X)

# Confusion table of in-built LDA function results
table(y, Y.pred$class)
```

```
##
## y     0   1
##   0 258  44
##   1  73  87
```

The above is the confusion table of predicted in-sample fitted labels using in-built LDA function.

```
# Comparing results from both LDA approaches

# Confusion table of results
table(y, pred_class)
```

```
##    pred_class
## y     0   1
##   0 258  44
##   1  73  87
```

```
# Confusion table of in-built LDA function results
table(y, Y.pred$class)
```

```
##
## y     0   1
##   0 258  44
##   1  73  87
```

Yes, the results match from both approaches as evidenced by same confusion table.

Question 2: QDA and Marginal Screening

From our lecture notes, we know that QDA does not work directly on the Hand Written Digit data. This is because the number of variables is larger than the number of observations for some class labels. Let's consider doing a small trick to this example, and see if that works. You should use the `zip.train` as the training data and `zip.test` as the testing data.

```
library(ElemStatLearn)

# Loading train data
X2 = zip.train[,-1]
Y2 = zip.train[, 1]

# Loading test data
X2_test = zip.test[,-1]
Y2_test = zip.test[, 1]
```

Instead of using all 256 variables, we will select 40 variables, and only use them to perform the QDA. The criteria for this selection is the marginal variance, meaning that we will calculate the variance of each variable in the training data, and pick the top 40 with the largest variance.

```
# Calculating marginal variance of all 256 variables in Hand Written Digit data matrix
variance = apply(X2, 2, var)
vars = c(1:256)

# Creating a data frame of variable and its corresponding variance
df_var = data.frame(vars, variance)
# Ordering the data frame with decreasing variance values
df_var = df_var[order(df_var$variance, decreasing = TRUE), ]

# Obtaining the top40 variables with largest variance
variables_top40_var = df_var[1:40, 1]
# Printing top 40 variables with largest variance
variables_top40_var
```

```
##  [1] 230 219 105 185 121 120 204  56 235 136 169 220 213 137  76 201  27  38  69
## [20]  22 200 104 214 153 152  53 107 123  60  54  57 234  92  91  75 139 154 184
## [39] 168  43
```

The 40 printed variables above have the largest variance in the training data.

Perform this analysis [20 Points] and report the testing data confusion table. Answer the following questions:

- [5 Points] Does the method work? Why do you think it works/or not?

```
# Performing QDA using all 256 variables
# dig.qda1 = qda(X2_test, Y2_test)
```

QDA method doesn't work when we include all 256 variates. Since we need to estimate the covariance matrix of each class, the number of observations for each class has to be at least larger than the number of variables(256 in this case) to make the covariance matrix invertible.

```
table(Y2_test)
```

```
## Y2_test
##   0   1   2   3   4   5   6   7   8   9
## 359 264 198 166 200 160 170 147 166 177
```

From above, we can realize that there are many classes [2 3 4 5 6 7 8 9] which have observations less than the number of variables (256 in out case), thus making the covariance matrix non-invertible. Hence, QDA doesn't work.

Now, trying QDA by reducing the number of variables.

```
# Creating data frame of test Hand Written Digit data matrix
Xtest_df = as.data.frame(X2_test)

# Subsetting data frame of Hand Written Digit data to contain only top 40 vars with largest variance
Xtest_40 = Xtest_df[, c(variables_top40_var)]
# Displaying observations of X_40 data frame
head(Xtest_40)
```

```
##     V230   V219   V105   V185   V121   V120   V204    V56 V235 V136   V169
## 1 -1.000 -0.932 -0.831  0.703  0.833  0.074 -1.000 -1.000   -1    1  0.155
## 2 -0.606  0.302 -1.000  1.000 -1.000 -1.000  0.976 -1.000   -1   -1  1.000
## 3  1.000  1.000  0.156 -1.000  1.000  1.000  0.943 -1.000    1    1 -1.000
## 4 -1.000 -0.648 -1.000 -0.215 -0.698 -1.000  0.896 -1.000   -1   -1  0.440
## 5  1.000  0.075 -1.000  1.000 -1.000 -1.000 -0.849  0.658    1   -1  0.878
## 6  1.000  1.000 -1.000 -1.000 -1.000 -1.000  0.657  0.446    1   -1 -1.000
##     V220   V213   V137    V76   V201    V27    V38    V69    V22   V200   V104
## 1 -1.000 -1.000  0.905  0.326  0.959  1.000  1.000  0.248  0.588 -0.833 -1.000
## 2 -0.522  1.000 -0.512 -1.000  1.000 -1.000  0.789  0.960 -0.783  0.998 -1.000
## 3  1.000 -0.333  1.000  0.930 -1.000  1.000 -0.150 -1.000  1.000 -1.000 -0.621
## 4 -0.780 -0.081  0.388 -1.000  1.000 -1.000 -0.417  1.000 -1.000  1.000 -1.000
## 5  0.833 -0.244 -1.000 -1.000  0.317 -0.388 -0.481  0.939 -1.000  1.000 -1.000
## 6  1.000  1.000 -1.000  0.941 -1.000 -0.831  0.498  0.960 -0.801 -1.000 -1.000
##     V214   V153   V152    V53   V107   V123    V60    V54    V57   V234    V92
## 1 -1.000 -0.237 -0.007  1.000  1.000  1.000 -0.390  0.010 -1.000  0.732  1.000
## 2  1.000  0.870 -0.643  0.963 -1.000 -0.136 -1.000  0.609 -1.000 -1.000 -1.000
## 3 -0.333 -0.333 -0.050 -0.524  1.000  0.952  0.344 -1.000 -1.000  1.000  1.000
## 4 -0.611  0.677 -1.000  0.447 -0.592  1.000 -1.000  0.999 -1.000 -1.000 -0.969
## 5 -0.661 -0.853 -1.000 -0.386 -1.000 -1.000 -1.000  0.913 -0.825  1.000 -1.000
## 6  0.868 -1.000 -1.000  0.070 -1.000 -1.000  1.000  1.000 -0.953  1.000 -0.015
##      V91    V75   V139   V154   V184   V168    V43
## 1  0.976 -0.402  1.000  1.000 -0.991 -1.000 -0.624
## 2 -1.000 -1.000  1.000  0.970  0.820  0.198 -1.000
## 3  0.821 -0.803  0.792 -0.172 -1.000 -1.000 -0.072
## 4 -1.000 -1.000  0.711  1.000 -0.215 -1.000 -1.000
## 5 -1.000 -1.000 -0.930 -0.172  0.999 -0.280 -1.000
## 6 -1.000 -0.238 -1.000 -1.000 -1.000 -1.000  0.926
```

```
# Performing QDA using only top40 variables with largest variance
dig.qda1 = qda(Xtest_40, Y2_test)
Y2.pred1 = predict(dig.qda1, Xtest_40)
```

Yes, the qda method works now. This is because number of observations in each class is greater than the number of parameters(40) causing to resolve the "some group is too small for 'qda' " error we had obtained with 256 parameters. Hence, the inverse exists with the reduced number of variables(40 now).

```
# Testing data confusion table
table(Y2_test, Y2.pred1$class)
```

```
##
## Y2_test   0   1   2   3   4   5   6   7   8   9
##       0 335   0  12   3   1   2   1   0   5   0
##       1   1 251   1   0   7   0   2   0   2   0
##       2   4   0 184   4   3   1   1   0   1   0
##       3   2   0   1 160   0   1   0   0   2   0
##       4   0   1   3   0 181   0   0   1   1  13
##       5   6   0   0   1   0 152   0   0   0   1
##       6   1   0   0   0   1   1 163   0   4   0
##       7   0   0   1   2   2   0   0 140   0   2
##       8   0   0   2   7   0   1   0   0 156   0
##       9   0   0   1   0   3   1   1   4   2 165
```

The testing data confusion table (with 40 variables).

- [5 Points] Decrease the number of variables that you select from 40 to just 10. What is the performance compared with the 40 variable version? Why do you think this happened?

```
# Obtaining the top10 variables with largest variance
variables_top10_var = df_var[1:10, 1]

# Subsetting data frame of Hand Written Digit data to contain only top 10 vars with largest variance
Xtest_10 = Xtest_df[, c(variables_top10_var)]
# Displaying observations of X_40 data frame
head(Xtest_10)
```

```
##     V230   V219   V105   V185   V121   V120   V204    V56 V235 V136
## 1 -1.000 -0.932 -0.831  0.703  0.833  0.074 -1.000 -1.000   -1    1
## 2 -0.606  0.302 -1.000  1.000 -1.000 -1.000  0.976 -1.000   -1   -1
## 3  1.000  1.000  0.156 -1.000  1.000  1.000  0.943 -1.000    1    1
## 4 -1.000 -0.648 -1.000 -0.215 -0.698 -1.000  0.896 -1.000   -1   -1
## 5  1.000  0.075 -1.000  1.000 -1.000 -1.000 -0.849  0.658    1   -1
## 6  1.000  1.000 -1.000 -1.000 -1.000 -1.000  0.657  0.446    1   -1
```

```
# Performing QDA using only top10 variables with largest variance
dig.qda2 = qda(Xtest_10, Y2_test)
Y2.pred2 = predict(dig.qda2, Xtest_10)

# Testing data confusion table
table(Y2_test, Y2.pred2$class)
```

```
##
## Y2_test   0   1   2   3   4   5   6   7   8   9
##       0 309   0  11   6   3   9  18   3   0   0
```

```
##        1   1 243   4   1   5   0   1   0   3   6
##        2  18   0  93  10  16   3  19  26   7   6
##        3   5   0   8 125   4  18   0   0   6   0
##        4   0   6   8   0  95   1   4  13  15  58
##        5  16   0   7  39   5  85   3   2   3   0
##        6  57   0  15   6   1   9  68   8   4   2
##        7   3   1   4   0   5   0   2 124   1   7
##        8   1   2   9  31  15   8   5   0  82  13
##        9   0   1   9   1  17   0   0  18   7 124
```

The testing data confusion table (with 10 variables).

```
# Comparing performance of both QDA versions

# Sum of elements along the diagonal for confusion table with 40 variables
sum(diag(table(Y2_test, Y2.pred1$class)))
```

```
## [1] 1887
```

```
# Sum of elements along the diagonal for confusion table with 10 variables
sum(diag(table(Y2_test, Y2.pred2$class)))
```

```
## [1] 1348
```

The elements along the diagonals represent correct prediction by the model. Since, both QDA versions have been fitted on same number of observations, the one with confusion table of higher trace(sum of elements along the diagonal) performs better.

As 1887>1348, we can conclude that the QDA with 40 variables makes more number of correct predictions and hence performs better than the QDA with 10 variables.

We attempted to select a subset of the variables by including the ones with high variance values so that it still contains most of the information in the full dataset. From the above results we can see that the performance with 40 variables is better than with 10 variables as a large proportion of the variance in the data is still explained by 40 components(with large variances) than with 10(with large variances). Hence, data with 40 components provides a better representation of full data leading to better results with QDA.