Problem Set 5: Discriminant Analysis

Stat 154, Fall 2017, Prof. Sanchez

Self grade due date: Th Nov-09 (before midnight)

Instructions

Do not give raw computer output as your main answer to any question. Remember that providing a clear and reasonable justification of your answers is at least as important as getting the right answer.

1) Sum-of-Squares Dispersion Functions (10 pts)

Function `tss()`: write a function `tss()` that computes the total sum of squares of a given variable:

$$\text{TSS} = \sum_{i=1}^{n}(x - \bar{x})^2$$

The function `tss()` should take one argument `x`, the input vector.

```
# total sum of squares
tss <- function(x) {
  sum((x - mean(x))^2)
}
```

When testing `tss()`, you should get the following output:

```
tss(iris$Sepal.Length)
```

```
## [1] 102.1683
```

Function `bss()`: write a function `bss()` that computes the between groups sum of squares:

$$\text{BSS} = \sum_{k=1}^{K} n_k(\bar{x}_k - \bar{x})^2$$

The function `bss()` takes two arguments:

- `x` = vector for the predictor variable
- `y` = vector (or factor) for the response variable
- *include a `stop()` statement if `x` and `y` have different lengths*

```r
# between-group sum of squares
bss <- function(x, y) {
  if (length(x) != length(y)) {
    stop('inputs have different length')
  }
  nk <- as.vector(table(y))
  x_means <- tapply(x, y, mean)
  sum(nk * ((x_means - mean(x))^2))
}
```

When testing `bss()`, you should get the following output:

```r
bss(iris$Sepal.Length, iris$Species)
```

```
## [1] 63.21213
```

Function `wss()`: write a function `wss()` that computes the wetween groups sum of squares:

$$\text{WSS} = \sum_{k=1}^{K} \sum_{i \in G_k} (x_{ik} - \bar{x}_k)^2$$

The function `wss()` takes two arguments:

- `x` = vector for the predictor variable
- `y` = vector (or factor) for the response variable
- *include a* ***`stop()`*** *statement if* ***`x`*** *and* ***`y`*** *have different lengths*

```r
# within-group sum of squares
wss <- function(x, y) {
  if (length(x) != length(y)) {
    stop('inputs have different length')
  }
  nk <- as.vector(table(y))
  ws <- tapply(x, y, function(u) sum((u - mean(u))^2))
  sum(ws)
}
```

When testing `wss()`, you should get the following output:

```r
wss(iris$Sepal.Length, iris$Species)
```

```
## [1] 38.9562
```

2) Sum-of-Squares Ratio Functions (10 pts)

Function `cor_ratio()`: use `bss()` and `tss()` to write a function `cor_ratio()` that computes the correlation ratio η^2 between a variable `x` and a response `y`.

$$\eta^2(x, y) = \frac{\text{BSS}}{\text{TSS}}$$

```
# correlation ratio
cor_ratio <- function(x, y) {
  bss(x, y) / tss(x)
}
```

Here's how you should be able to call `cor_ratio()`

```
cor_ratio(iris$Sepal.Length, iris$Species)
```

```
## [1] 0.6187057
```

Function `F_ratio()`: use `bss()` and `tss()` to write a function `F_ratio()` that computes the F-ratio between a variable `x` and a response `y`.

$$F = \frac{\text{BSS}/(k-1)}{\text{WSS}/(n-k)}$$

```
# F-ratio
F_ratio <- function(x, y) {
  y <- as.factor(y)
  n <- length(x)
  k <- nlevels(y)
  (bss(x, y) / (k - 1)) / (wss(x, y) / (n - k))
}
```

Here's how you should be able to call `F_ratio()`

```
F_ratio(iris$Sepal.Length, iris$Species)
```

```
## [1] 119.2645
```

---

3) Discriminant Power of Predictors (30 pts)

For this part of the assignment, consider wines of classes 1 and 2. The idea is to rank the predictors using three approches: simple logistic regressions, correlation ratios, and F-ratios.

```r
# use classes 1 and 2
wine <- read.csv('wine.data')
wine12 <- wine[wine$class != 3, ]
wine12$class[wine12$class == 2] <- 0
```

Simple logistic regressions (10 pts)

Run simple logistic regressions for each predictor and the response, and store the values of the AIC statistic.

```r
# simple logistic regressions
var_labels <- names(wine12)[-1]
AIC <- rep(0, length(var_labels))

for (j in 1:length(var_labels)) {
  lg <- glm(wine12$class ~ wine12[ ,var_labels[j]], family = binomial)
  AIC[j] <- lg$aic
}
```

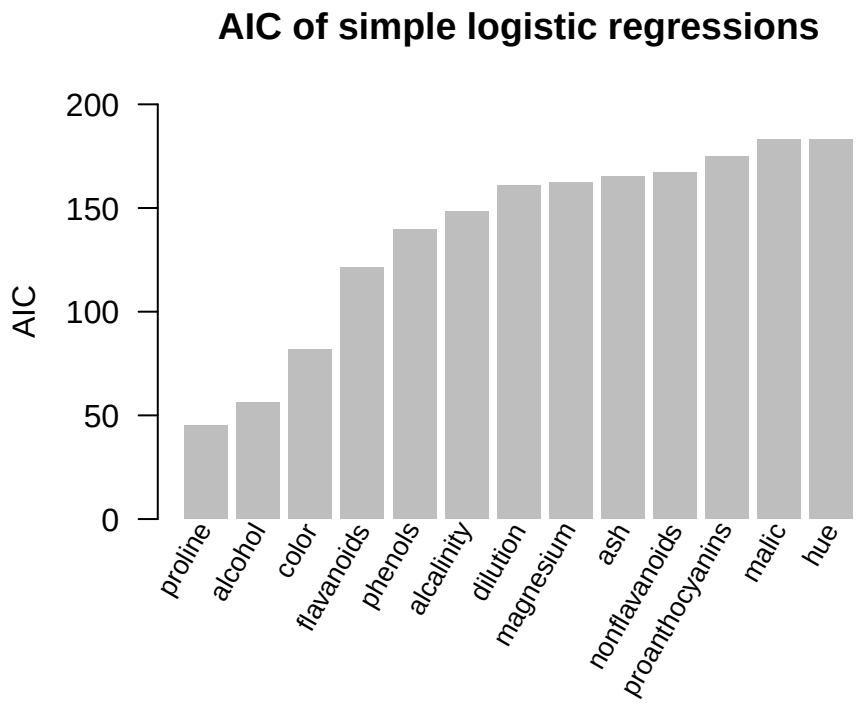
Make a table (e.g. data frame) with the predictors ranked by AIC value in increasing order. The smallest the AIC, the more discriminant the predictor.

```r
aic_tbl <- data.frame(
  variable = var_labels[order(AIC)],
  AIC = AIC[order(AIC)],
  stringsAsFactors = FALSE
)
aic_tbl
```

```
##           variable       AIC
## 1          proline  45.21948
## 2          alcohol  56.30075
## 3            color  81.96971
## 4       flavanoids 121.51589
## 5          phenols 139.62520
## 6       alcalinity 148.51462
## 7         dilution 161.00793
## 8        magnesium 162.10222
## 9              ash 165.30370
## 10   nonflavanoids 166.94370
## 11 proanthocyanins 174.71983
## 12           malic 182.85454
## 13             hue 183.07125
```

Display the AICs in a barchart.

```r
bp_aic <- barplot(aic_tbl$AIC, border = NA,
        las = 1, ylim = c(0, 200), ylab = "AIC")
text(bp_aic, -3, labels = aic_tbl$variable, srt = 60,
     xpd = TRUE, adj = 1, cex = 0.8)
title("AIC of simple logistic regressions")
```



Correlation ratios (10 pts)

Calculate correlation ratios for each predictor and the response.

```r
# correlation ratios
ETA <- rep(0, length(var_labels))

for (j in 1:length(var_labels)) {
  ETA[j] <- cor_ratio(wine12[ ,var_labels[j]], wine12$class)
}
```

Make a table (e.g. data frame) with the predictors ranked by η^2 value in increasing order. The largest the η^2, the more discriminant the predictor.

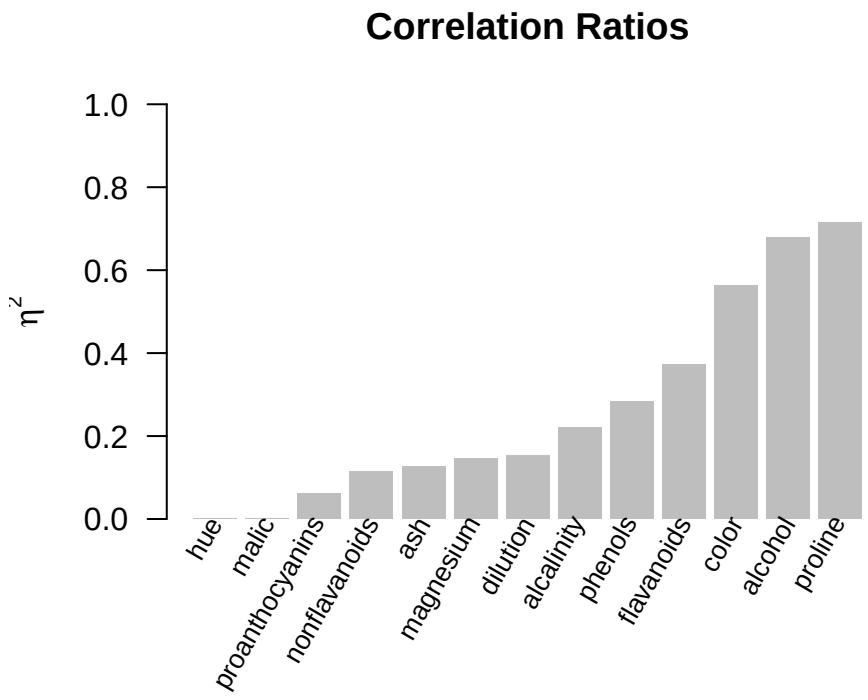
```r
eta_tbl <- data.frame(
  variable = var_labels[order(ETA)],
  ETA = ETA[order(ETA)],
  stringsAsFactors = FALSE
)
```

```
eta_tbl
```

```
##           variable          ETA
## 1              hue 0.0002904483
## 2            malic 0.0019626987
## 3  proanthocyanins 0.0621031600
## 4    nonflavanoids 0.1138987546
## 5              ash 0.1257044543
## 6        magnesium 0.1467544634
## 7         dilution 0.1534649761
## 8       alcalinity 0.2213110717
## 9          phenols 0.2837609465
## 10      flavanoids 0.3729916215
## 11           color 0.5634196215
## 12         alcohol 0.6796337087
## 13         proline 0.7145258216
```

Display the η^2's in a barchart.

```
bp_eta <- barplot(eta_tbl$ETA, border = NA,
                  las = 1, ylim = c(0, 1),
                  ylab = expression(eta^2))
text(bp_eta, 0, labels = eta_tbl$variable, srt = 60,
     xpd = TRUE, adj = 1, cex = 0.8)
title("Correlation Ratios")
```



F-ratios (10 pts)

Calculate F-ratios for each predictor and the response.

```
# F ratios
Frs <- rep(0, length(var_labels))

for (j in 1:length(var_labels)) {
  Frs[j] <- F_ratio(wine12[ ,var_labels[j]], wine12$class)
}
```

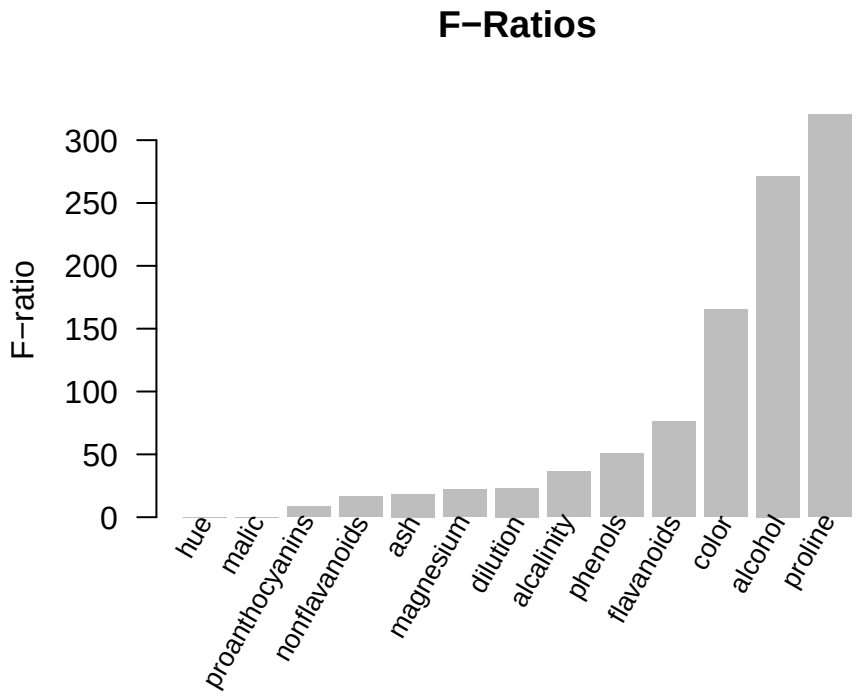
Make a table (e.g. data frame) with the predictors ranked by F-value in increasing order. The largest the F, the more discriminant the predictor.

```
F_tbl <- data.frame(
  variable = var_labels[order(Frs)],
  Fratio = Frs[order(Frs)],
  stringsAsFactors = FALSE
)
F_tbl
```

```
##           variable       Fratio
## 1              hue   0.03718818
## 2            malic   0.25171949
## 3  proanthocyanins   8.47556377
## 4    nonflavanoids  16.45301895
## 5              ash  18.40358243
## 6        magnesium  22.01543461
## 7         dilution  23.20461220
## 8       alcalinity  36.37886215
## 9          phenols  50.71128273
## 10      flavanoids  76.14400251
## 11           color 165.18770681
## 12         alcohol 271.54265938
## 13         proline 320.37680494
```

Display the F-values in a barchart.

```
bp_fr <- barplot(F_tbl$Fratio, border = NA,
                 las = 1, ylim = c(0, 330),
                 ylab = "F-ratio")
text(bp_fr, 0, labels = F_tbl$variable, srt = 60,
     xpd = TRUE, adj = 1, cex = 0.8)
title("F-Ratios")
```



4) Variance functions

Function `total_variance()` (10 pts)

Write a function `total_variance()` that takes a matrix of predictors, and returns the (sample) variance-covariance matrix $\mathbf{V}$. Do NOT use `var()` to create `total_variance()`.

```
# total variance
total_variance <- function(X) {
  n <- nrow(X)
  X <- as.matrix(scale(X, scale = FALSE))
  (1/(n-1)) * t(X) %*% X
}
```

Here's how yo should be able to invoke `total_variance()`, and compare it with the `var()` function.

```
# test total_variance()
total_variance(iris[ ,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

```r
# compare with var()
var(iris[ ,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

Function `between_variance()` (10 pts)

Write a function `between_variance()` that takes a matrix of predictors, and a response vector (or factor), and returns the (sample) Between-variance matrix $\mathbf{B}$. Do NOT use `var()` to create `between_variance()`.

```r
# between variance
between_variance <- function(X, y) {
  X <- as.matrix(X)
  y <- as.factor(y)
  K <- nlevels(y)
  levs <- levels(y)
  nk <- as.vector(table(y))
  g <- colMeans(X)

  B <- 0
  for (k in 1:K) {
    gk <- colMeans(X[y == levs[k], ])
    B <-  B + (nk[k]) * (gk - g) %*% t(gk - g)
  }
  rownames(B) <- colnames(X)
  (1/(nrow(X)-1)) * B
}
```

Here's how yo should be able to invoke `between_variance()` on `iris` data

```r
# test between_variance()
between_variance(iris[ ,1:4], iris$Species)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.4242425 -0.13391051    1.1090497   0.4783848
## Sepal.Width    -0.1339105  0.07614049   -0.3841584  -0.1539105
## Petal.Length    1.1090497 -0.38415839    2.9335758   1.2535168
## Petal.Width     0.4783848 -0.15391051    1.2535168   0.5396868
```

Function `within_variance()` (10 pts)

Write a function `within_variance()` that takes a matrix of predictors, and a response vector (or factor), and returns the (sample) Within-variance matrix $\mathbf{W}$. Do NOT use `var()` to create `within_variance()`.

```
# within variance
within_variance <- function(X, y) {
  X <- as.matrix(X)
  y <- as.factor(y)
  K <- nlevels(y)
  levs <- levels(y)
  nk <- as.vector(table(y))
  g <- colMeans(X)

  W <- 0
  for (k in 1:K) {
    Xk <- scale(X[y == levs[k], ], scale = FALSE)
    W <- W + t(Xk) %*% Xk
  }
  (1/(nrow(X)-1)) * W
}
```

Here's how yo should be able to invoke `within_variance()` on `iris` data (10 pts)

```
# test within_variance()
within_variance(iris[ ,1:4], iris$Species)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length   0.26145101  0.09147651   0.16526577  0.03788591
## Sepal.Width    0.09147651  0.11383893   0.05450201  0.03227114
## Petal.Length   0.16526577  0.05450201   0.18270201  0.04209262
## Petal.Width    0.03788591  0.03227114   0.04209262  0.04131946
```

Confirm that $\mathbf{V} = \mathbf{B} + \mathbf{W}$

```
# confirm V = B + W
Viris <- total_variance(iris[ ,1:4])
Viris
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

```r
# B + W
Biris <- between_variance(iris[ ,1:4], iris$Species)
Wiris <- within_variance(iris[ ,1:4], iris$Species)
Biris + Wiris
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

Challenge (70 pts)

Use the predictors and response of the wine data, to write code in R that allows you to find the eigenvectors $\mathbf{u_k}$. (20 pts)

```r
# predictors, response, and indices
X <- as.matrix(wine[ ,-1])
y <- as.factor(wine$class)
k <- nlevels(y)
n <- nrow(X)
nk <- as.vector(table(y))

# matrix of group means (i.e. centroids)
centroids <- matrix(0, ncol(X), k)
for (j in 1:ncol(X)) {
  centroids[j,] <- tapply(X[,j], y, FUN=mean)
}
dimnames(centroids) <- list(colnames(X), levels(y))
centroids
```

```
##                             1          2           3
## alcohol            13.744746  12.278732  13.1537500
## malic               2.010678   1.932676   3.3337500
## ash                 2.455593   2.244789   2.4370833
## alcalinity         17.037288  20.238028  21.4166667
## magnesium         106.338983  94.549296  99.3125000
## phenols             2.840169   2.258873   1.6787500
## flavanoids          2.982373   2.080845   0.7814583
## nonflavanoids       0.290000   0.363662   0.4475000
## proanthocyanins     1.899322   1.630282   1.1535417
## color               5.528305   3.086620   7.3962500
## hue                 1.062034   1.056282   0.6827083
```

```
## dilution           3.157797   2.785352   1.6835417
## proline         1115.711864 519.507042 629.8958333
```

```
# decomposing between-class matrix:  B = CC'
gm <- colMeans(X)
centroids_centered <- sweep(centroids, 1, gm, FUN="-")
C <- sweep(centroids_centered, 2, sqrt(nk/n), FUN="*")
C
```

```
##                             1             2            3
## alcohol           0.42841387 -4.559188e-01   0.07952005
## malic            -0.18749696 -2.549459e-01   0.51794151
## ash               0.05128360 -7.687942e-02   0.03664452
## alcalinity       -1.41493680  4.693073e-01   0.99793297
## magnesium         3.79830190 -3.279269e+00  -0.22281367
## phenols           0.31380368 -2.288742e-02  -0.32007129
## flavanoids        0.54872651  3.257331e-02  -0.64797693
## nonflavanoids    -0.04136819  1.141897e-03   0.04447521
## proanthocyanins   0.17756730  2.487287e-02  -0.22711557
## color             0.27071522 -1.245115e+00   1.21418500
## hue               0.06021202  6.241915e-02  -0.14267052
## dilution          0.31441054  1.096821e-01  -0.48197649
## proline         212.33853989 -1.436095e+02 -60.75568494
```

```
# within-groups covariance matrix
W <- within_variance(X, y)
```

Thus, we can diagonalize (i.e. EVD) the following symmetric matrix:

$$\mathbf{C}^{\mathsf{T}}\mathbf{W}^{-1}\mathbf{C}$$

and then use the eigenvector $\mathbf{w}$ to recover $\mathbf{u}$

$$\mathbf{u} = \mathbf{W}^{-1}\mathbf{C}\mathbf{w}$$

```
# eigen-decomposition
EIG <- eigen(t(C) %*% solve(W) %*% C)
lam <- EIG$values[1:(k - 1)]
U <- solve(W) %*% C %*% EIG$vectors[,1:(k-1)]

head(U)
```

```
##                   [,1]          [,2]
## alcohol     1.219170420  1.7764466510
## malic      -0.499438832  0.6222701484
```

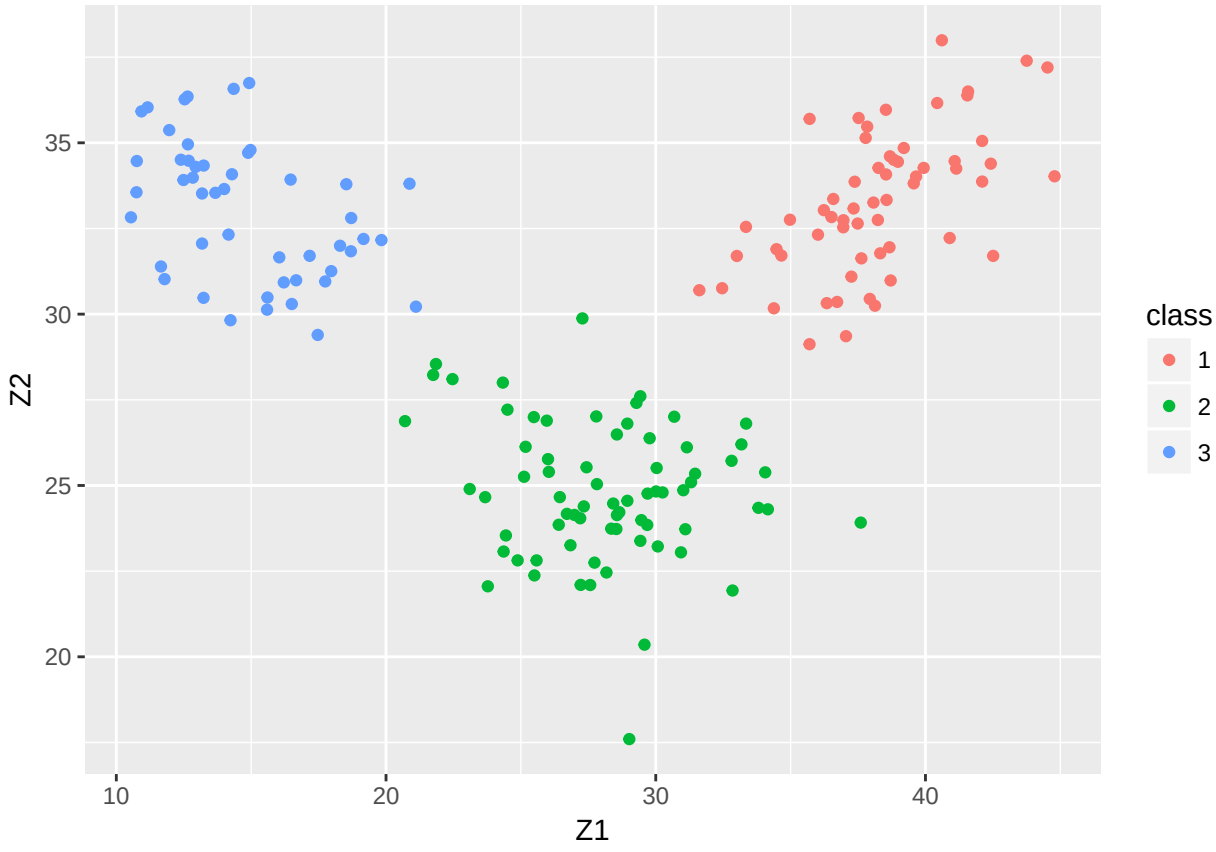
```
## ash          1.115433516  4.7801216521
## alcalinity -0.467836167 -0.2982790636
## magnesium   0.006538602 -0.0009429557
## phenols    -1.867900866 -0.0656398326
```

Obtain the linear combinations $\mathbf{z_k}$ and make a scatterplot of the wines. Add color to the dots indicating the different classes. (10 pts)

It is possible that the scale of your scatterplot is different, or even that the shape is different (e.g. you may have an inverted image of my plot). The important thing is the relative position of the cloud of points.

```r
# canonical scores (components)
Z <- as.data.frame(X %*% U)
names(Z) <- c("Z1", "Z2")
Z$class <- y

# scatterplot (canonical variables)
ggplot(data = Z, aes(x = Z1, y = Z2, color = class)) +
  geom_point()
```

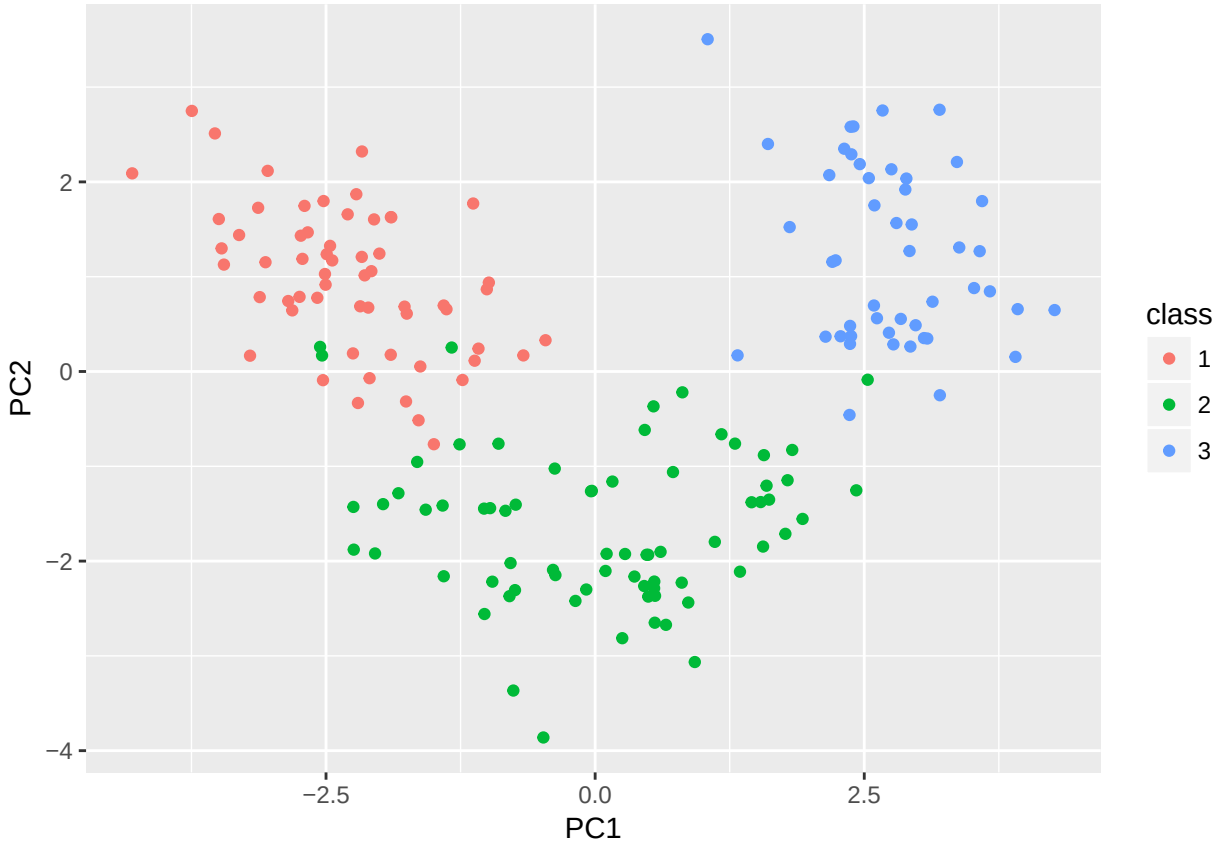


Obtain a scatterplot of the wines but this time using the first two principal components on the standardized predictors. Add color to the dots indicating the different classes. How does

this compare to the previous scatterplot? (10 pts)

```
# principal components analysis (PCA)
pca <- prcomp(X, scale. = TRUE)
PC <- as.data.frame(pca$x)
PC$class <- y

# PCA scatterplot
ggplot(data = PC, aes(x = PC1, y = PC2, color = class)) +
  geom_point()
```



Notice that the PCA plot does not provide a class separation as good as the one obtained in canonical discriminant analysis (CDA). Although the wine classes seem to be fairly differentiated, there are ares with overlapped dots between classes 1 and 2, and between classes 2 and 3.

Calculate the correlations between $\mathbf{z_k}$ and the predictors. How do you interpret each score? (10 pts)

```
# correlations between predictors and CDA components
cor(X, Z[ ,1:2])
```

```
##                          Z1          Z2
```

```
## alcohol          0.27989693  0.81621795
## malic           -0.48917598  0.31781550
## ash              0.01918243  0.40451247
## alcalinity      -0.52999779 -0.21482152
## magnesium        0.19359267  0.33551963
## phenols          0.75482118  0.07008972
## flavanoids       0.89849357 -0.02635971
## nonflavanoids   -0.51522117 -0.02507846
## proanthocyanins  0.53203867 -0.05042644
## color           -0.34411332  0.76652309
## hue              0.68407589 -0.37803538
## dilution         0.85037786 -0.20319881
## proline          0.61489470  0.67171319
```

Create a matrix of size $n \times K$, with the squared Mahalanobis distances $d^2(\mathbf{x_i}, \mathbf{g_k})$ of each observation $\mathbf{x_i}$ (i.e. each wine) to the each of the k centroids $\mathbf{g_k}$. (10 pts)

```
# Mahalanobis (squared) distances
Winv <- solve(W)
Mahalanobis <- matrix(0, n, k)
for (i in 1:n) {
  for (h in 1:k) {
    dis2centroid <- (X[i, ] - centroids[,h])
    Mahalanobis[i,h] <- t(dis2centroid) %*% Winv %*% (dis2centroid)
  }
}
head(Mahalanobis)
```

```
##           [,1]     [,2]     [,3]
## [1,] 11.471872 51.37512 92.28077
## [2,]  8.738074 39.13556 83.11946
## [3,]  7.884262 34.50203 68.51471
## [4,] 13.484011 67.09116 87.00835
## [5,] 11.668097 17.12809 42.12974
## [6,]  6.913637 55.98424 85.16075
```

Finally, assign each observation to the class G_k for which the Mahalanobis distance $d^2(\mathbf{x_i}, \mathbf{g_k})$ is the smallest. And create a confussion matrix comparing the actual `class` versus the predicted class. (10 pts)

```
# predict minimum Mahalanobis distance
pred_class <- apply(Mahalanobis, 1, which.min)

table(pred_class)
```

```
## pred_class
```

```
##  1  2  3
## 59 71 48
```

```
# confusion matrix
table(obs = wine$class, pred = pred_class)
```

```
##    pred
## obs  1  2  3
##   1 59  0  0
##   2  0 71  0
##   3  0  0 48
```