Linear Discriminant Analysis

Martin Endler endlemar@fel.cvut.cz

Introduction The aim of this assignment is to get familiar with Linear Discriminant Analysis (LDA). LDA and Principal Component Analysis (PCA) are two techniques for dimensionality reduction. PCA can be described as an unsupervised algorithm that ignores data labels and aims to find directions which maximalize the variance in a data. In comparison with PCA, LDA is a supervised algorithm and aims to project a dataset onto a lower dimensional space with good class separability. In other words, LDA maximalizes the ratio of between-class variance and the within-class variance in a given data.

Input data In this tutorial, we will work with a dataset that classifies wines (samples) into three classes using of 13 continuous attributes; for more details see wine info.txt file. The dataset is located in wine.csv.

Linear Discriminant Analysis As we mentioned above, LDA finds directions where classes are well-separated, i.e. LDA maximizes the ratio of between-class variance and the within-class variance. Firstly, assume that C is a set of classes and set D, which represents a training dataset, is defined as $D = \{x_1, x_2, ..., x_N\}$.

The between-classes scatter matrix S_B is defined as:

$S_B = \sum_c N_C(\mu_c - \overline{x})(\mu_c - \overline{x})^T$,

where $\overline{x}$ is a vector represents the overall mean of the data, μ_c represents the mean corresponding to each class, and N_C are sizes of the respective classes.

The within-classes scatter matrix S_W is defined as:

$S_W = \sum_c \sum_{x \in D_c} (x - \overline{\mu_c})(x - \overline{\mu_c})^T$

Next, we will solve the generalized eigenvalue problem for the matrix $S_W^{-1} S_B$ to obtain the linear discriminants, i.e.

$(S_W^{-1} S_B)w = \lambda w$

where w represents an eigenvector and λ represents an eigenvalue. Finally, choose k eigenvectors with the largest eigenvalue and transform the samples onto the new subspace.

Step by step

Load the dataset

```
mydata <- read.csv("wine.csv", header = FALSE)
labels <- mydata[, 1]
labels <- as.factor(labels)
mydata <- mydata[, -1]
```

```
ComputeWithinScatter <- function(data, num_variables) {

    S_W <- matrix(0, nrow = num_variables, ncol = num_variables)
```

```
    for (class in seq_along(data)) {
        D_c <- as.matrix(data[[class]])
        mean_c <- colMeans(D_c)
        D_c_minus_mean <- sweep(D_c, 2, mean_c)
        S_W_c <- t(D_c_minus_mean) %*% D_c_minus_mean
        S_W <- S_W + S_W_c
    }

    return(S_W)
}
```

Compute the within-scatter matrix

```
ComputeBetweenScatter <- function(data, num_variables, mean_overall) {

    S_B <- matrix(0, nrow = num_variables, ncol = num_variables)

    for (class in seq_along(data)) {
        D_c <- as.matrix(data[[class]])
        N_c <- dim(D_c)[1]
        mean_c <- colMeans(D_c)
        mean_c_minus_overall <- mean_c - mean_overall
        S_B_c <- mean_c_minus_overall %*% t(mean_c_minus_overall)
        S_B <- S_B + (N_c * S_B_c)
    }

    return(S_B)
}
```

Compute the between-scatter matrix

```
SolveEigenProblem <- function(withinMatrix, betweenMatrix) {
    eivectors <- eigen(solve(withinMatrix) %*% betweenMatrix)
    return(eivectors)
}
```

Solve the EigenProblem and return eigen-vector

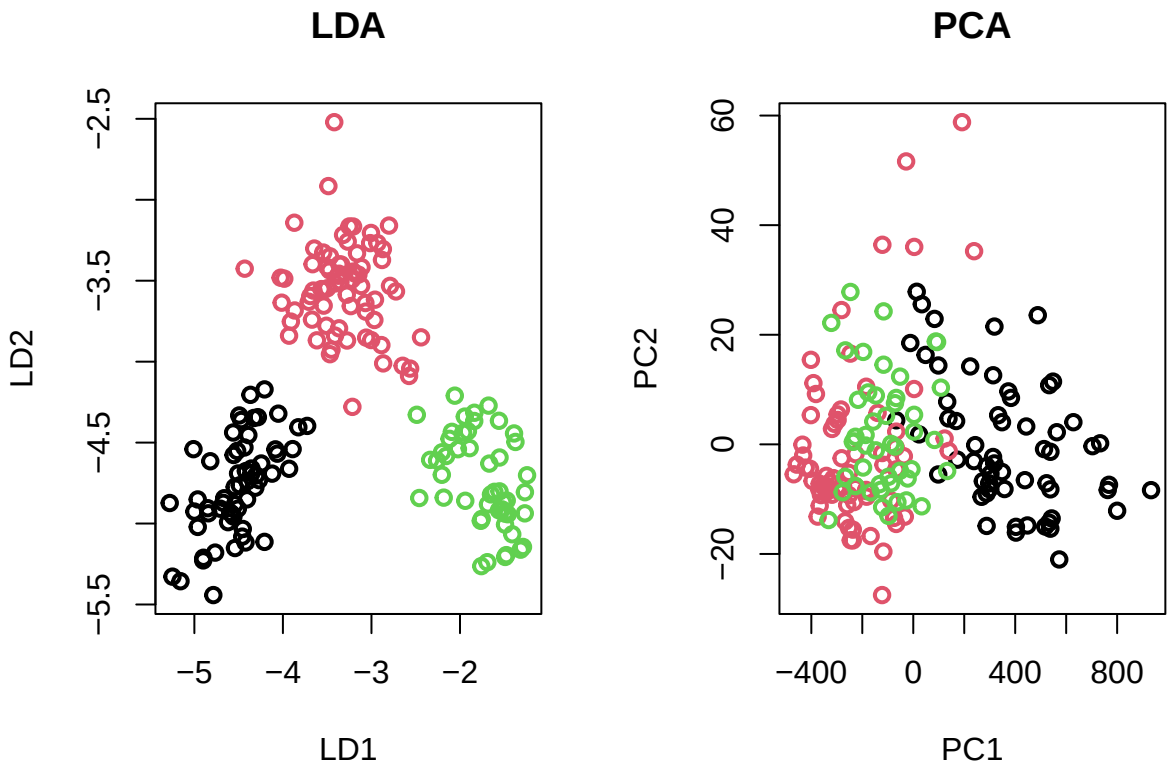
Visualize the results Project your data into lower-dimensional subspace, visualize this projection, and compare with PCA (see Fig. 1). Also, try to use scale/unscale version of `prcomp` function in R. Use the following code while filling in the lines marked as `TODO`.

```
ComputeCentroids <- function(data, labels) {
    yGroupedMean <- aggregate(as.data.frame(data), by = list(labels), FUN = mean)
    rownames(yGroupedMean) <- yGroupedMean[, 1]
    yGroupedMean <- yGroupedMean[, -1]
    return(yGroupedMean)
}

Classify <- function(newData, eigenVectors, labels, centroids) {
    y <- as.matrix(newData) %*% eigenVectors[, 1:(length(levels(labels)) - 1)]
    prior <- table(labels) / sum(table(labels))
```

```
    classification <- matrix(nrow = nrow(newData), ncol = length(levels(labels)))
    colnames(classification) <- levels(labels)
    for (c in levels(labels)) {
        classification[, c] <- as.matrix(
            0.5 * rowSums((y - matrix(
                rep(
                    as.matrix(centroids[c,]),
                    nrow(newData)
                ),
                nrow = nrow(newData),
                byrow = TRUE
            ))^2) - log(prior[c])
        )
    }
    return(levels(labels)[apply(classification, MARGIN = 1, which.min)])
}

CrossvalidationLDA <- function(mydata, labels, kfolds = 10) {
    set.seed(17)
    # randomly shuffle the data
    random <- sample(nrow(mydata))
    data <- mydata[random,]
    labels <- labels[random]
    # create 10 equally size folds
    folds <- cut(seq(1, nrow(data)), breaks = kfolds, labels = FALSE)
    acc <- rep(0, times = kfolds)
    # 10 fold cross validation
    for (i in 1:kfolds) {
        # segment your data by fold using the which() function
        testIndexes <- which(folds == i, arr.ind = TRUE)
        testData <- data[testIndexes,]
        trainData <- data[-testIndexes,]
        testLabels <- labels[testIndexes]
        trainLabels <- labels[-testIndexes]

        eigenLDA <- LDA(trainData, trainLabels)
        centroids <- ComputeCentroids(
            as.matrix(trainData) %*% eigenLDA[, 1:(length(levels(trainLabels)) - 1)],
            labels = trainLabels
        )
        pre <- Classify(
            newData = testData,
            labels = trainLabels,
            eigenVectors = eigenLDA,
            centroids = centroids
        )
        acc[i] <- sum(pre == testLabels) / length(testLabels)
    }
    return(mean(acc))
}

LDA <- function(mydata, labels) {
```

```
    # number of classes
    n <- length(levels(labels))
    num_variables <- dim(mydata)[2]

    # 1) split the data w.r.t. given factors
    splittedData <- split(mydata, labels)

    # 2) scatter matrices
    #############  within-class scatter matrix S_W ##################
    withinScatterMatrix <- ComputeWithinScatter(
        data = splittedData,
        num_variables = num_variables
    )
    #############  between-class scatter matrix S_B ##################
    betweenScatterMatrix <- ComputeBetweenScatter(
        data = splittedData,
        num_variables = num_variables,
        mean_overall = colMeans(mydata)
    )

    # 3)  eigen problem
    ############ solve Eigen problem #################################
    ei <- SolveEigenProblem(withinScatterMatrix, betweenScatterMatrix)

    # transform the samples onto the new subspace
    y <- (as.matrix(mydata) %*% ei$vectors[, 1:2])

    ## visual comparison with PCA
    par(mfrow = c(1, 2))
    pca <- prcomp(mydata, scale. = FALSE) # bad
    # pca <- prcomp(mydata, scale. = TRUE) # good
    plot(y[, 1], y[, 2], col = labels, pch = 21, lwd = 2, xlab = "LD1", ylab = "LD2", main = "LDA")
    plot(-pca$x, col = labels, pch = 21, lwd = 2, main = "PCA")

    return(ei$vectors)
}

############################## FUNCTIONS END ###################################


############################## MAIN ############################################

### PREPARE DATA
# data(iris)
# mydata <- iris
# labels <- mydata[,5]
# mydata <- mydata[,-5]

# already done in Section `##### Load the dataset`
# mydata <- read.csv("wine.csv", header = FALSE)
# labels <- mydata[, 1]
# labels <- as.factor(labels)
# mydata <- mydata[, -1]
```

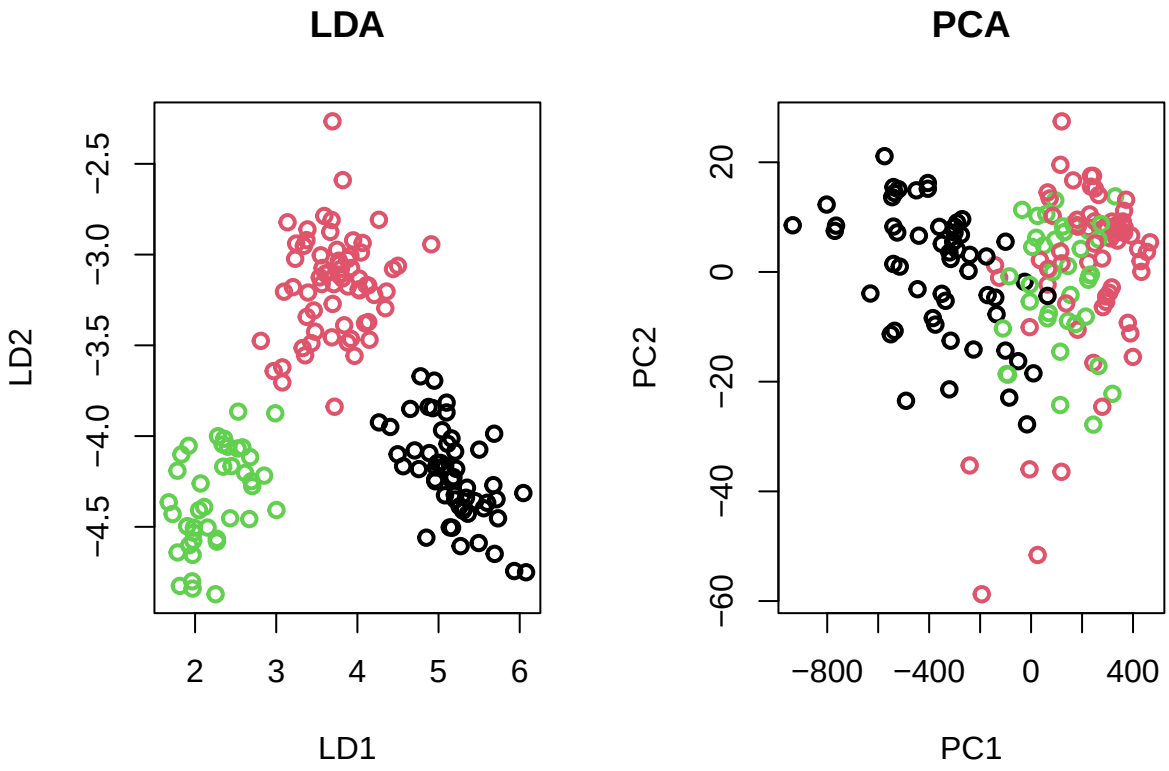
```
# compute LDA and return corresponding eigenvectors
eigenLDA <- LDA(mydata, labels)
```

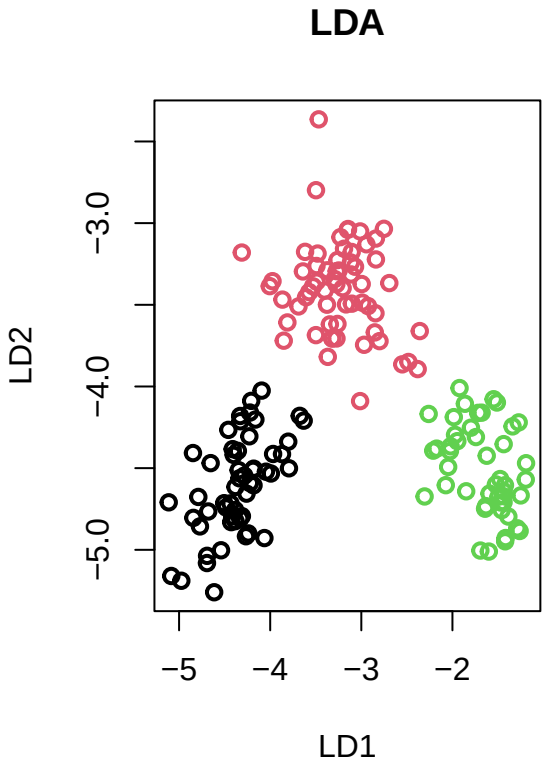


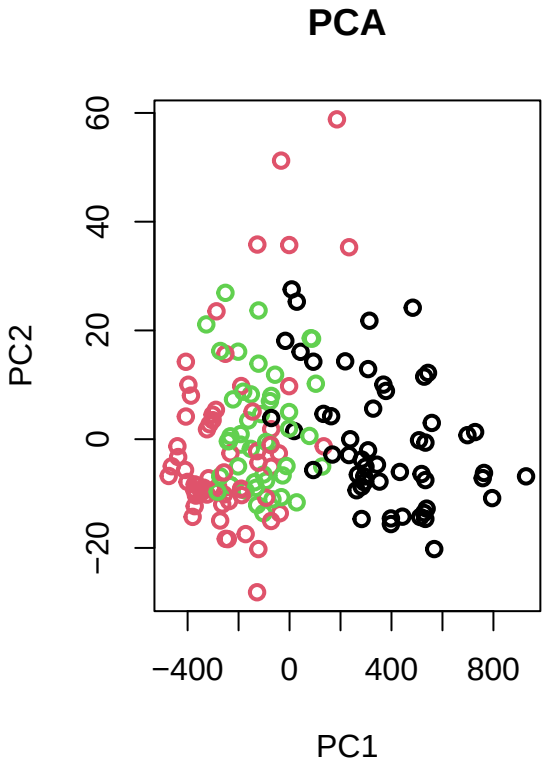

```
# find centroids in the transformed data
centroids <- ComputeCentroids(
    as.matrix(mydata) %*% eigenLDA[, 1:(length(levels(labels)) - 1)],
    labels = labels
)
# make predictions on the "mydata"
prediction <- Classify(
    newData = mydata,
    labels = labels,
    eigenVectors = eigenLDA,
    centroids = centroids
)
# ACC
sum(prediction == labels) / (length(labels))
```

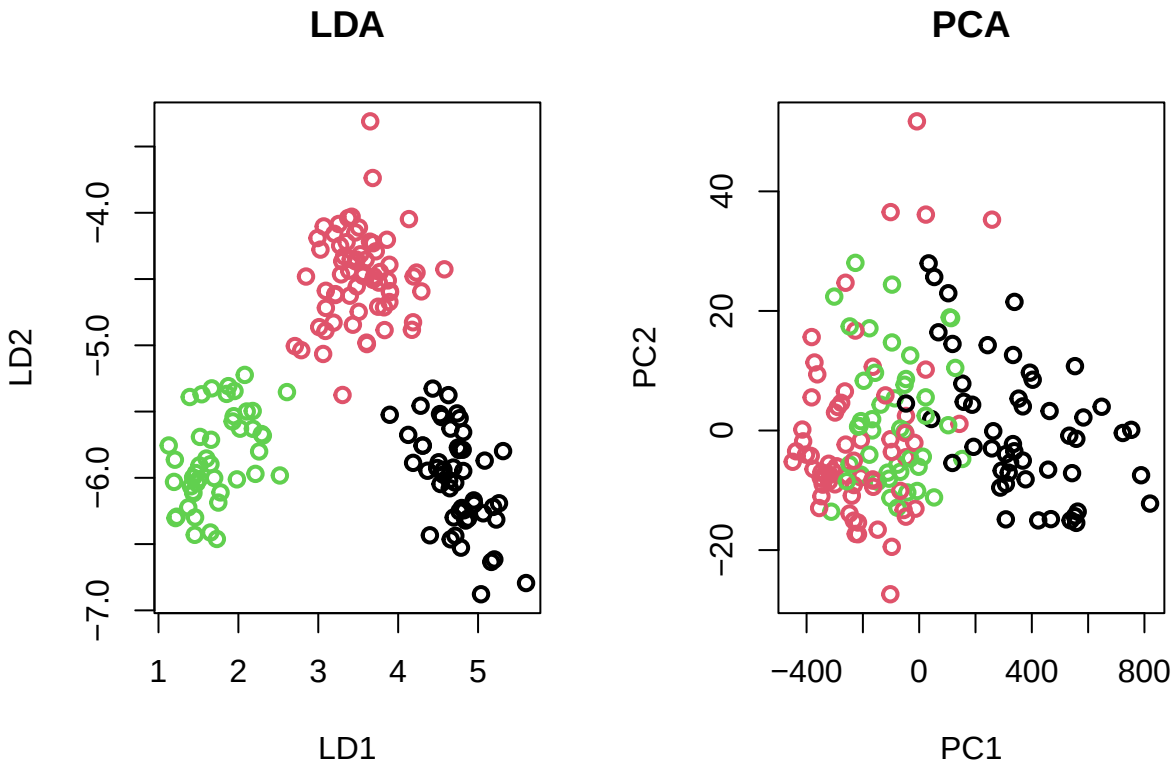
```
## [1] 0.988764
```

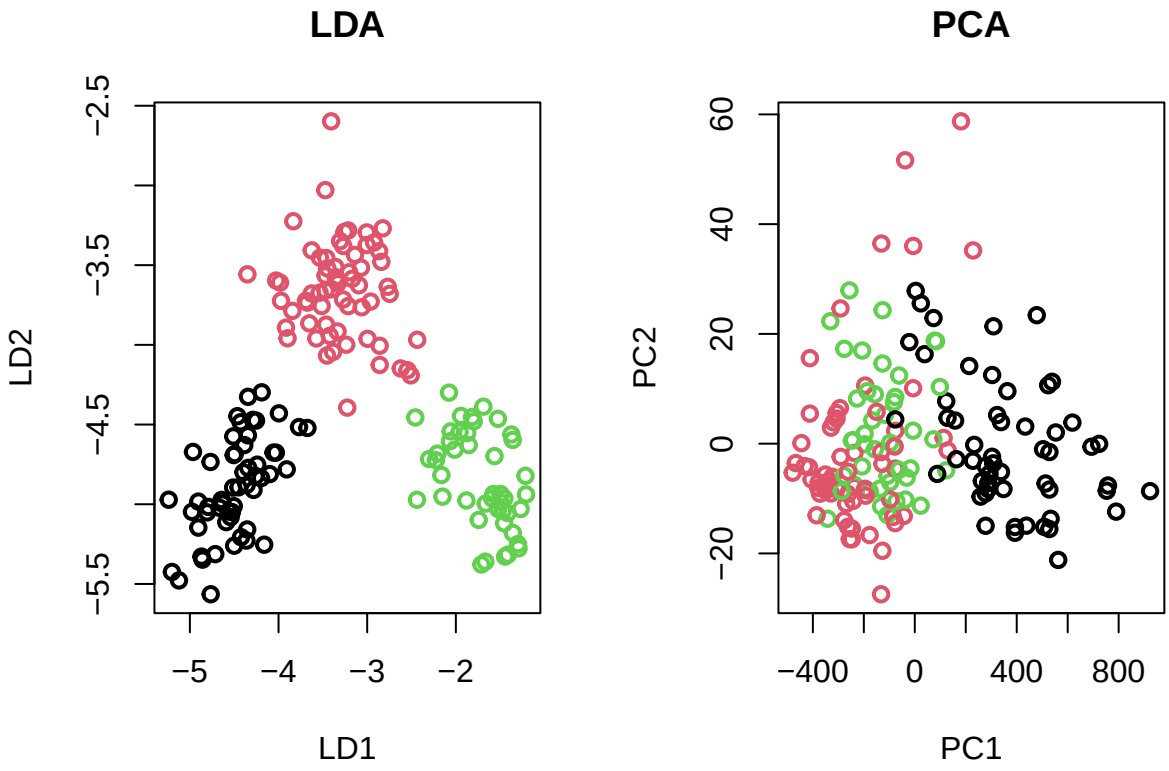
```
# CrossValidation
accLDA <- CrossvalidationLDA(mydata, labels, kfolds = 10)
```

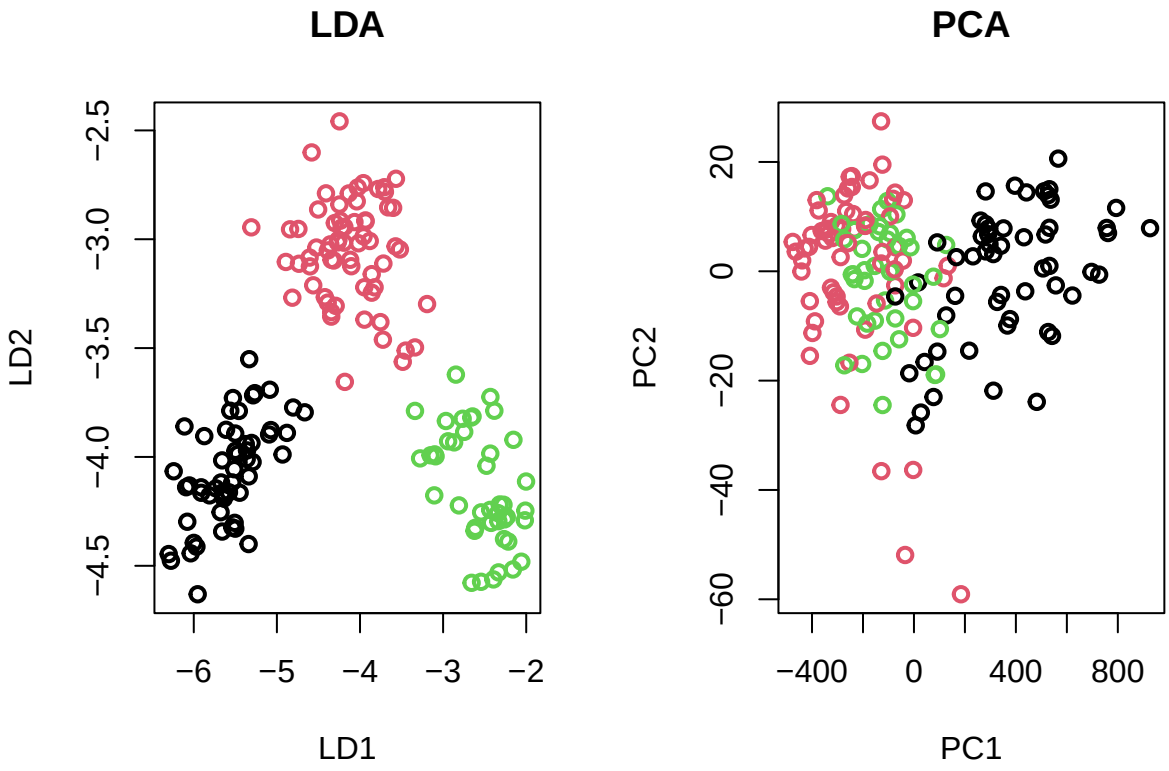

LDA
LD2
−2.5
−3.0
−3.5
−4.0
−4.5
2
3
4
5
6
LD1
PCA
PC2
20
0
−20
−40
−60
−800
−400
0
400
PC1

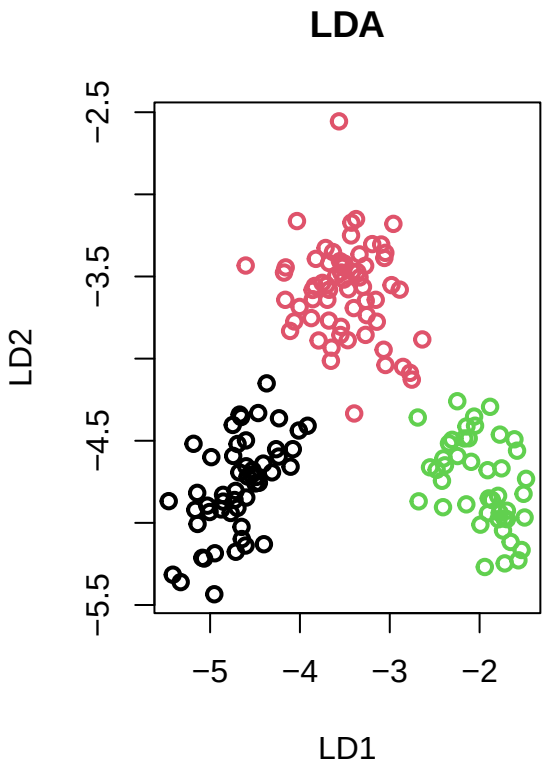

LDA
LD2
−3.0
−4.0
−5.0
−5
−4
−3
−2
LD1

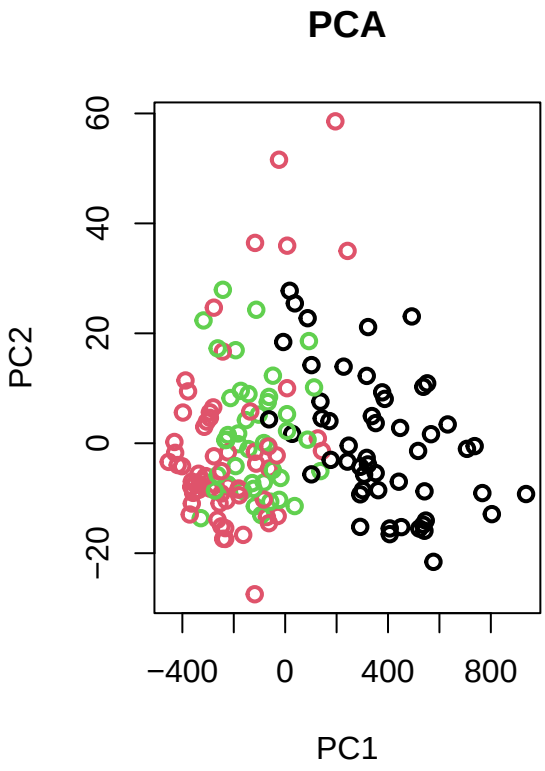

PCA
PC2
60
40
20
0
−20
−400
0
400
800
PC1

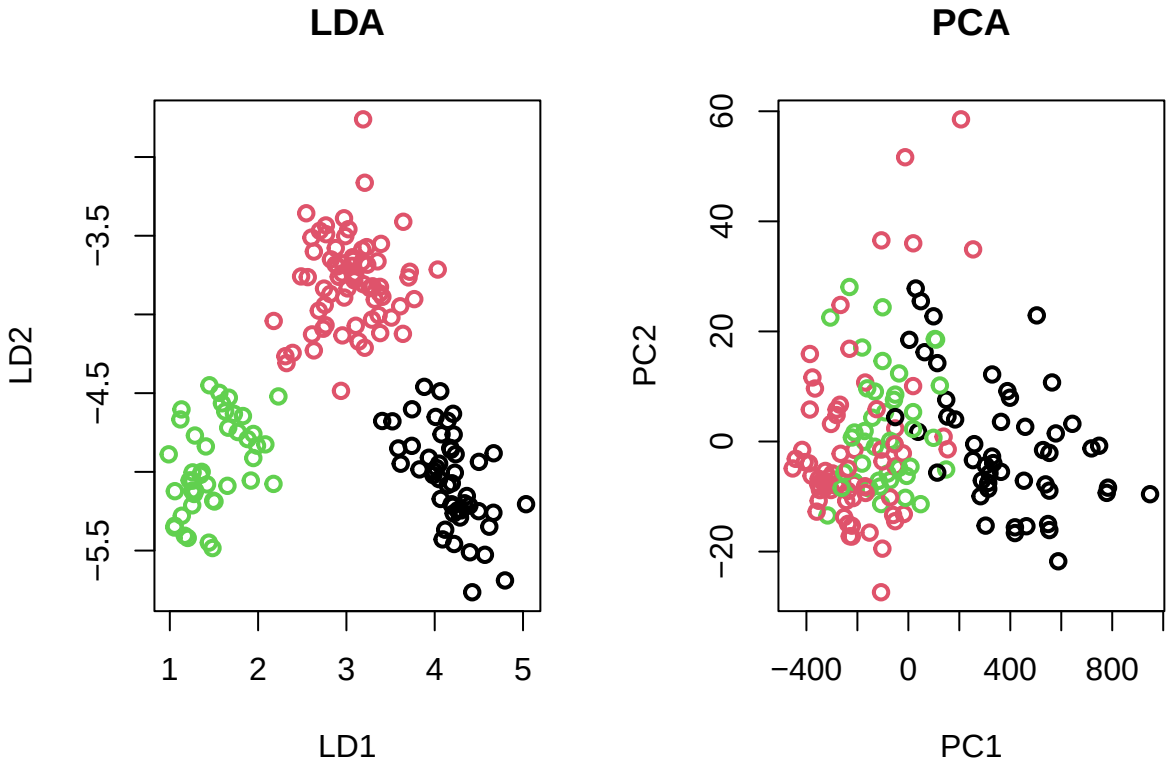

LDA
LD2
−4.0
−5.0
−6.0
−7.0
1
2
3
4
5
LD1
PCA
PC2
40
20
0
−20
−400
0
400
800
PC1

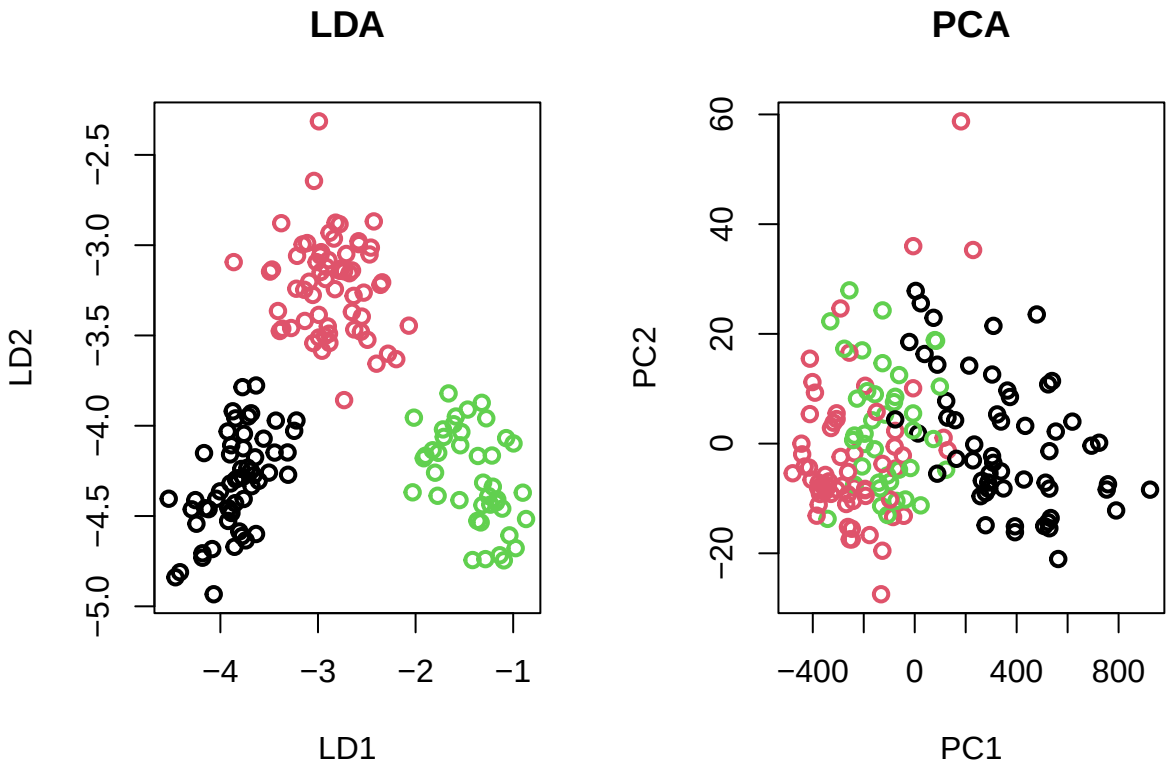

LDA
LD2
−2.5
−3.5
−4.5
−5.5
−5
−4
−3
−2
LD1
PCA
PC2
60
40
20
0
−20
−400
0
400
800
PC1

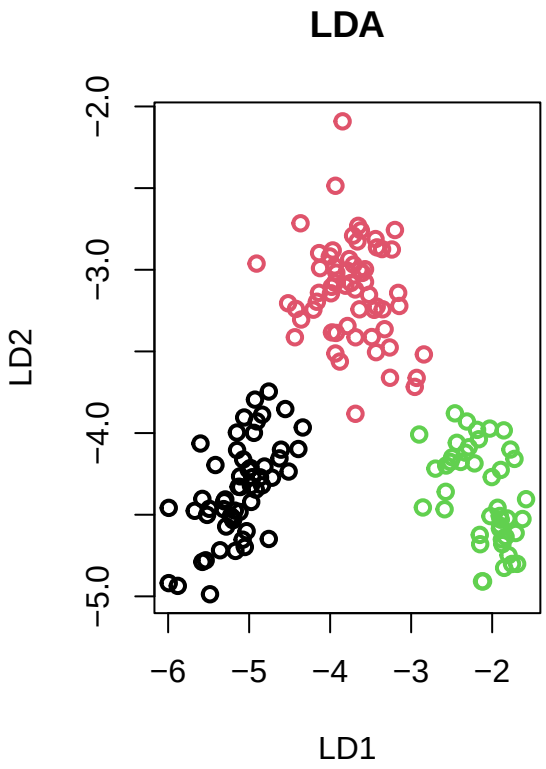

LDA
LD2
−2.5
−3.0
−3.5
−4.0
−4.5
−6
−5
−4
−3
−2
LD1
PCA
PC2
20
0
−20
−40
−60
−400
0
400
800
PC1

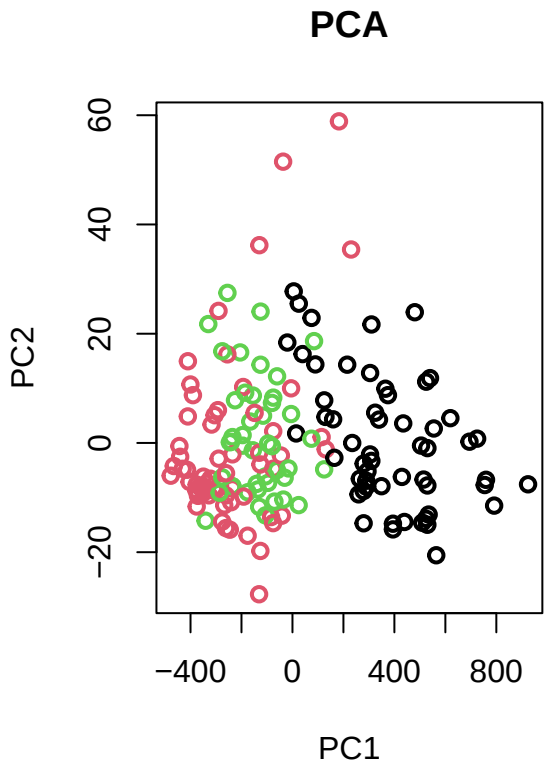

LDA
LD2
−2.5
−3.5
−4.5
−5.5
−5
−4
−3
−2
LD1

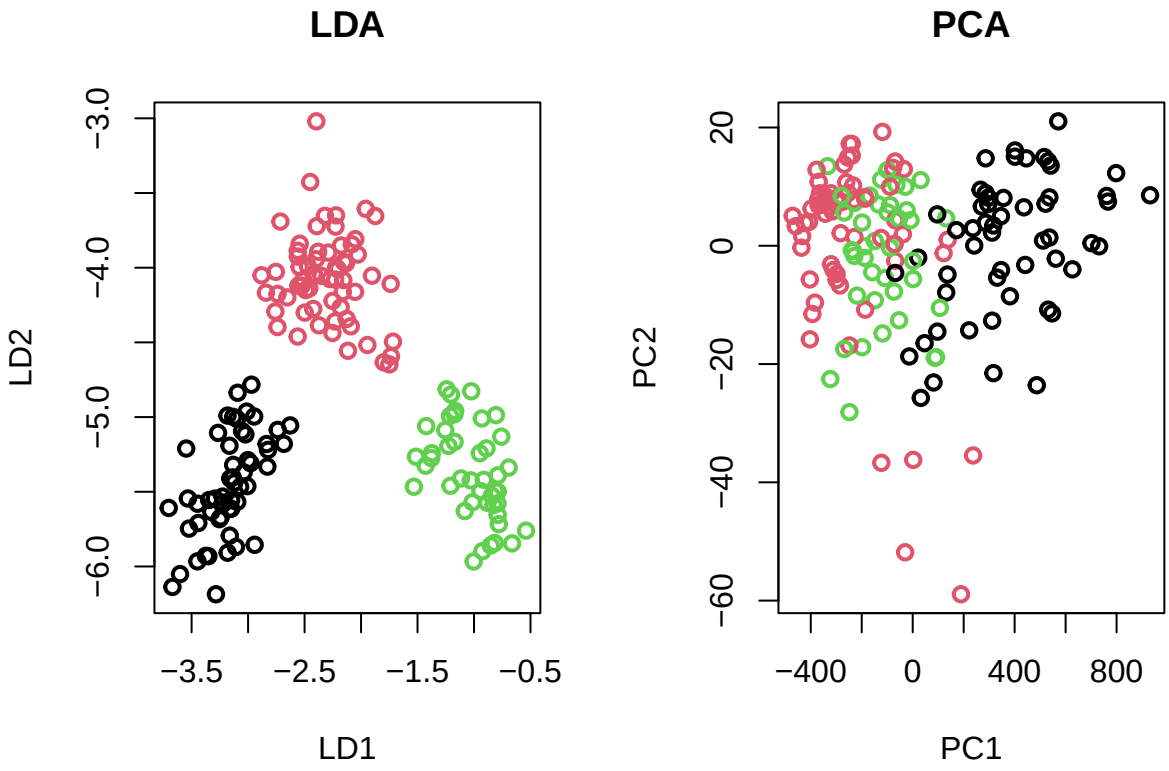

PCA
PC2
60
40
20
0
−20
−400
0
400
800
PC1


LDA
LD2
−3.5
−4.5
−5.5
1
2
3
4
5
LD1
PCA
PC2
60
40
20
0
−20
−400
0
400
800
PC1


LDA
LD2
−5.0
−4.5
−4.0
−3.5
−3.0
−2.5
−4
−3
−2
−1
LD1
PCA
PC2
−20
0
20
40
60
−400
0
400
800
PC1


LDA
LD2
−2.0
−3.0
−4.0
−5.0
−6
−5
−4
−3
−2
LD1


PCA
PC2
60
40
20
0
−20
−400
0
400
800
PC1



Discuss given results. **LDA was able to separate the given data nearly perfectly**. When trained on the whole dataset, only two 2 data points out of 178 were misclassified (i.e., ACC = 0.988764). Cross-validation on 10 random folds also confirmed this nearly-perfect performance of LDA on this dataset (mean ACC = 0.9888889).

Compared to the LDA, **PCA (without scaling)** performed very **bad** which was clearly visible from the plots. The reason for this is that there are big differences in the magnitude of variance between some variables (specifically variable 13 has a variance of several orders of magnitude larger than the rest). And because PCA aims to find directions which maximalize the variance in a data, this variable significantly affects the result. However, if we pass `.scale = TRUE` to `prcomp`, the variables will be scaled to have unit variance before the (PCA) analysis takes place. When we performed **PCA with scaling**, we got much better results *(compared to the PCA without scaling)*, but there were still **more misclassifications** than in the LDA.

```
# see that variable 13 is significantly affecting PCA without scaling
# compute the covarinace matrix
mydata.cov <- cov(mydata)
# see the variances of all variables (i.e., diagonal of the covarinace matrix)
print(mydata.cov[col(mydata.cov) == row(mydata.cov)])
# see the covarinace of variable 13
print(mydata.cov[13, 13])
# means are also useful
mydata.means <- colMeans(mydata)
print(mydata.means[13])
```