Linear Discriminanat Analysis

Diem Huong Nguyen

November 4, 2019

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 decribed 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.

The deadline of this assignment is November 25.

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 at 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 SB 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 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

```
# Skript je předělán tak, aby pracoval se splitnutými daty
# jako je uvedeno v MAINu. Spusťte tedy prosím load v mainu.

# Zde je zakomentovaný vlastní dataset load
#wineData <- read.csv(file = "wine.csv")
#labels <- wineData[,1]
#labels <- as.factor(labels)
#wineData <- wineData[,-1]
```

Compute the within-scatter matrix

```
ComputeWithinScatter <- function(data, n)
{
  # Covariance matrix
  covMatrix <- lapply(data,cov)
  numRow <- as.vector(unlist(lapply(data,nrow)))
  withinMatrix <- Reduce('+',mapply(function(x,y) x*(y-1),covMatrix,numRow, SIMPLIFY=FALSE))

  return(withinMatrix)
}
```

Compute the between-scatter matrix

```
ComputeBetweenScatter <- function(data, n, meanOverall)
{
  # Compute class mean
  classMean <- by(mydata, labels, colMeans)
  # make it a matrix instead of 3 lists in a matrix
  classMean <- matrix(unlist(classMean), ncol = ncol(mydata), byrow = TRUE)

  numRow <- as.vector(unlist(lapply(data,nrow)))
  diff <- list()
  for (i in 1:n) {
    diff[[i]] <- numRow[i] *
      (classMean[i,] - meanOverall) %*% t(classMean[i,] - meanOverall)
  }
  betweenMatrix <- Reduce('+',diff)
  return(betweenMatrix)
}
```

Solve the EigenProblem and return eigen-vector

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

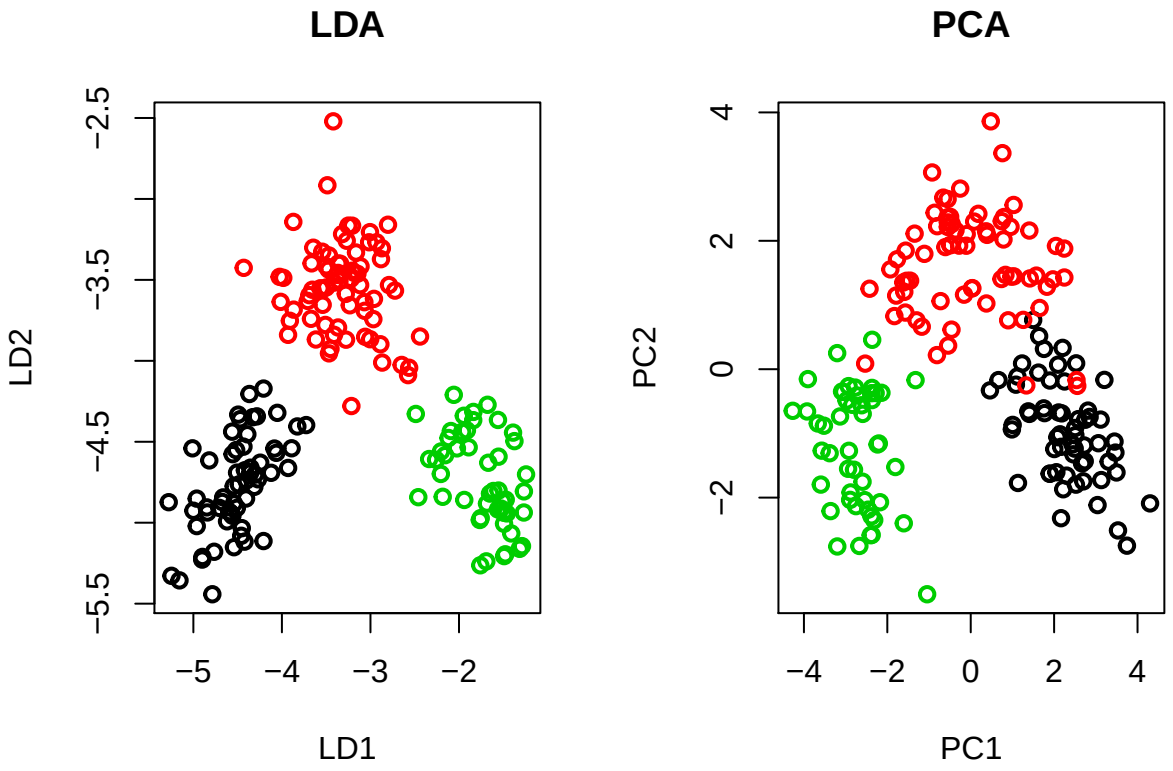
Visualize the results

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))

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

  # 2) scatter matrices
  #############  within-class scatter matrix Sw ##################
  withinScatterMatrix <- ComputeWithinScatter(splittedData, n)
```

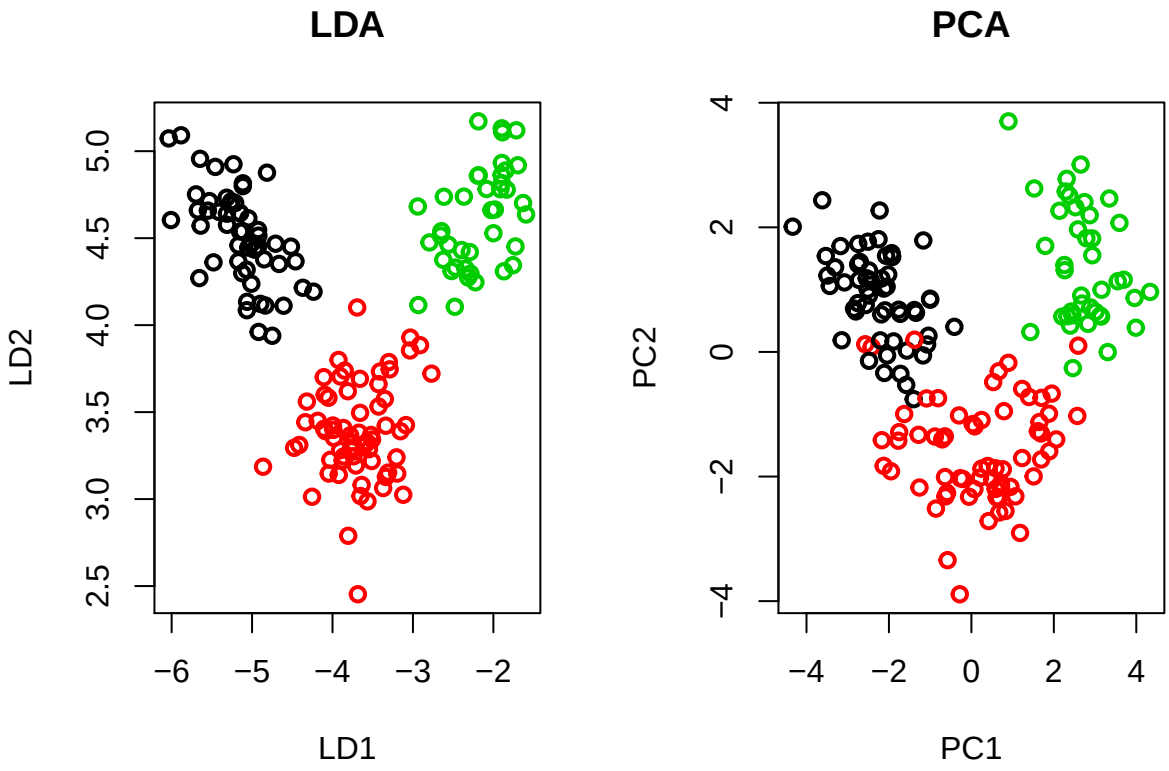
```
  ##############  between-class scatter matrix Sb ##################
  betweenScatterMatrix <- ComputeBetweenScatter(splittedData, n, colMeans(mydata))

  # 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. = TRUE)
  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]

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)
```

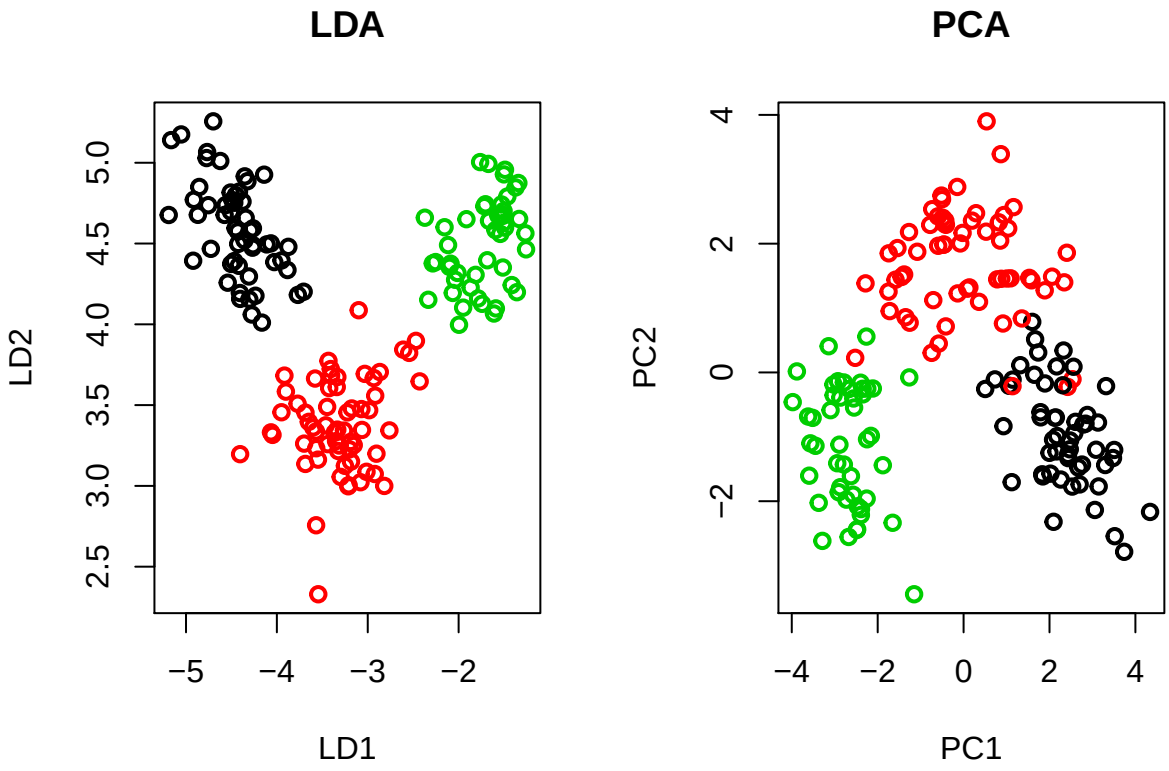


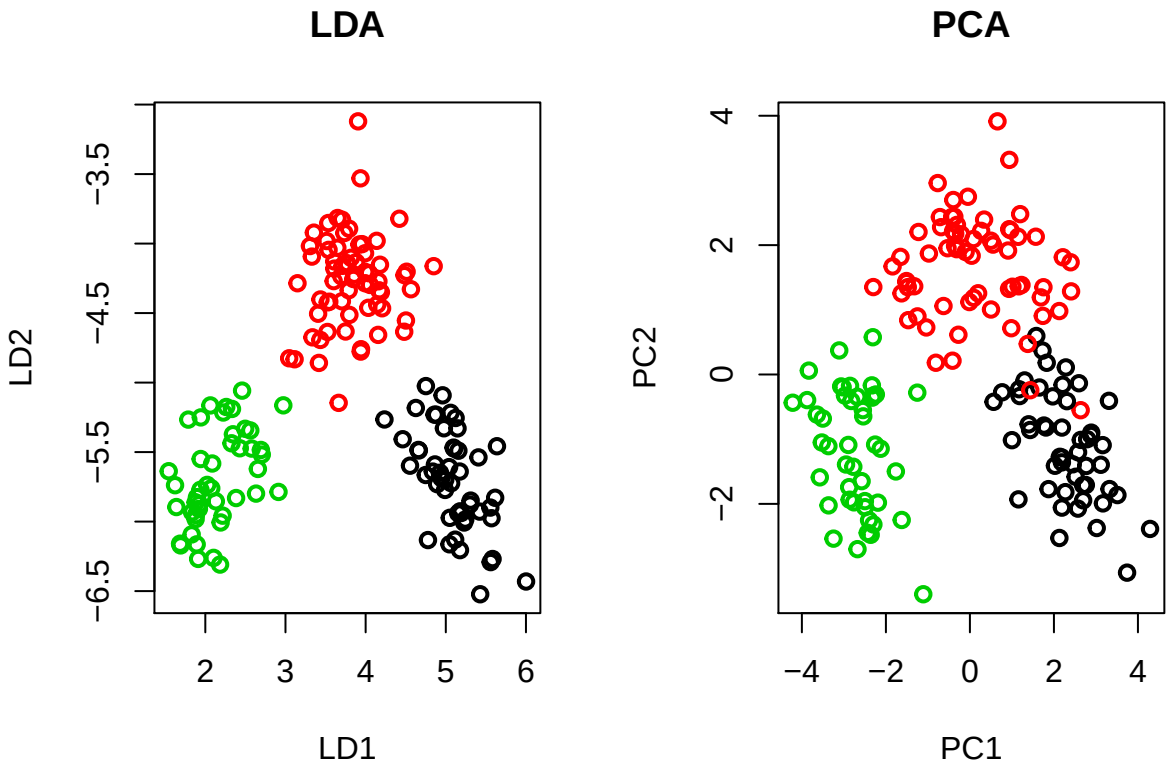

```
#find centroids in the transformed data
centroids <- ComputeCentroids(as.matrix(mydata) %*% eigenLDA[,1:(length(levels(labels))-1)],
                              labels = labels)
#make predictions on the "mydata"
prediction <- Classify(newData = mydata, labels = labels, eigenVectors = eigenLDA,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

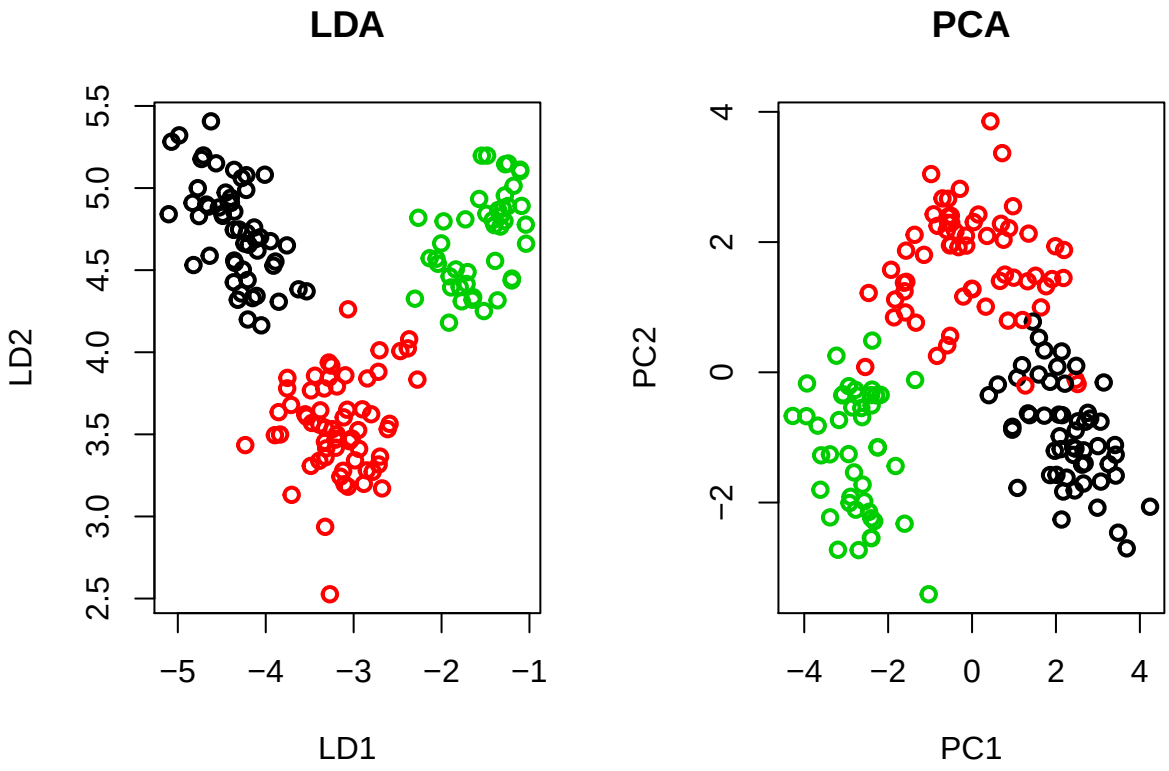
```
## [1] 0.988764
```

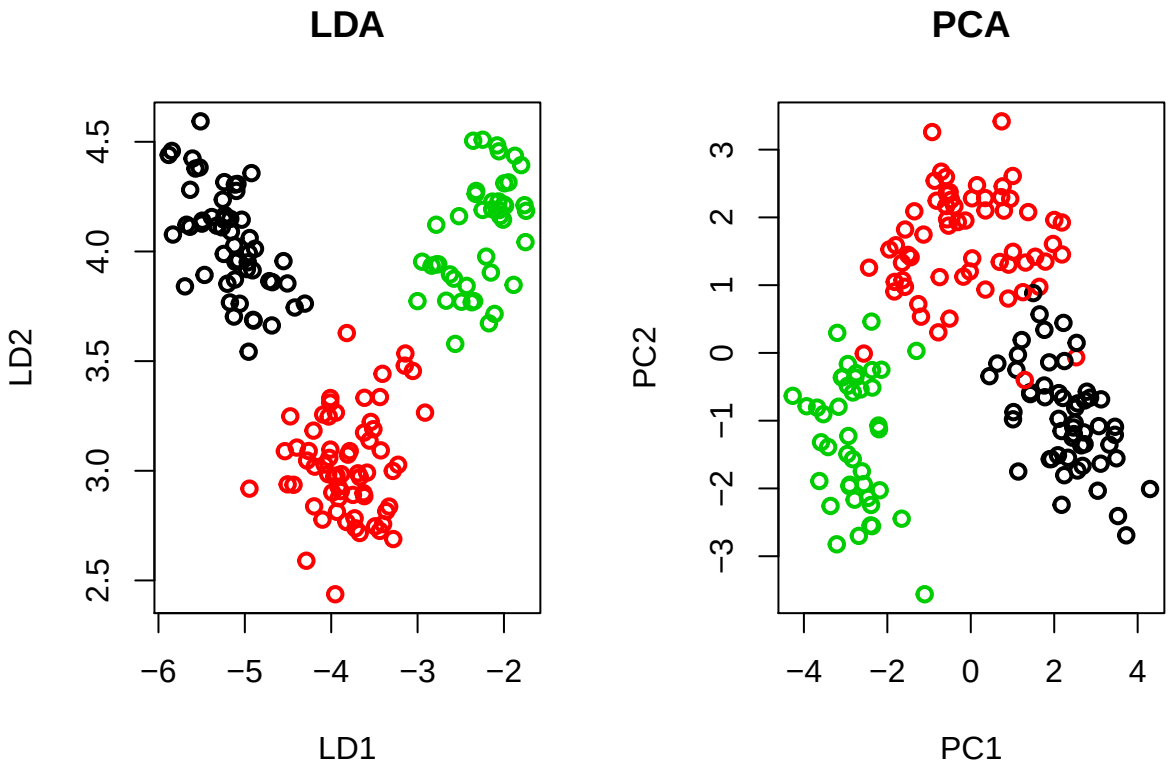
```
#CrossValidation
accLDA <- CrossvalidationLDA(mydata, labels, kfolds = 10)
```

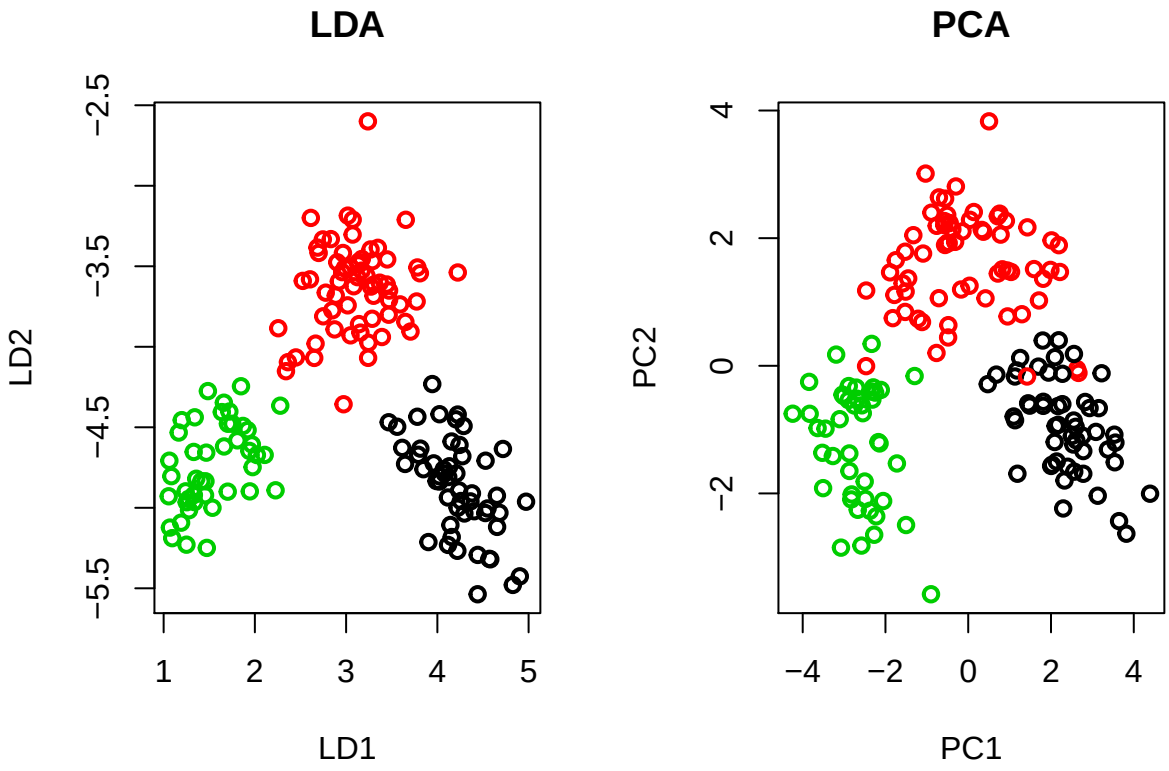

LDA
LD2
LD1
5.0
4.5
4.0
3.5
3.0
2.5
−6
−5
−4
−3
−2
PCA
PC2
PC1
4
2
0
−2
−4
−4
−2
0
2
4

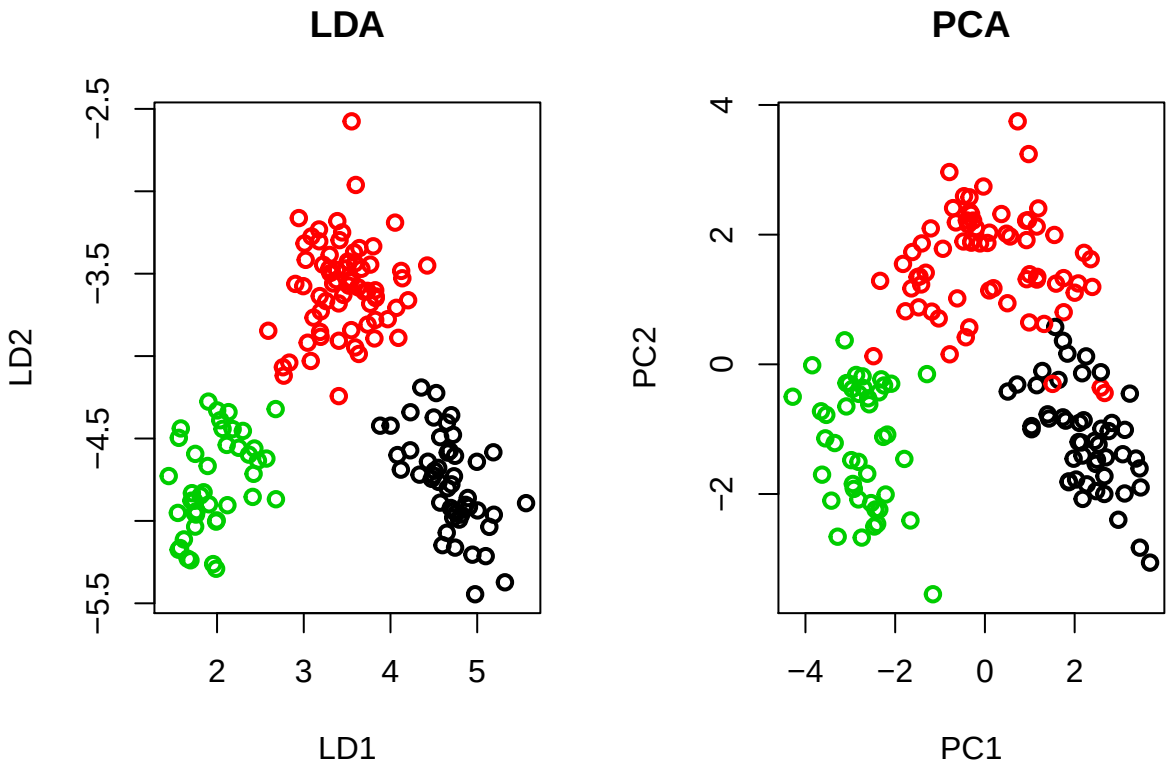

LDA
LD2
LD1
2.5
3.0
3.5
4.0
4.5
5.0
−5
−4
−3
−2
PCA
PC2
PC1
−2
0
2
4
−4
−2
0
2
4

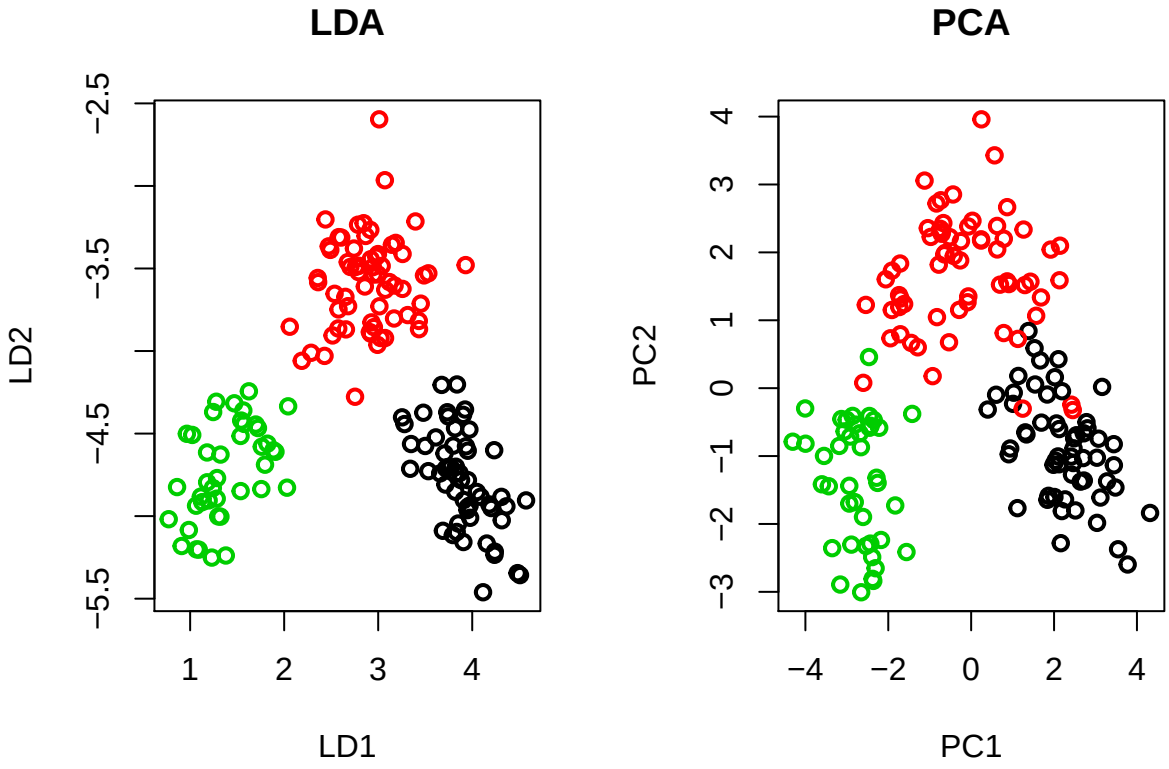

LDA
LD2
−3.5
−4.5
−5.5
−6.5
2
3
4
5
6
LD1
PCA
PC2
4
2
0
−2
−4
−2
0
2
4
PC1

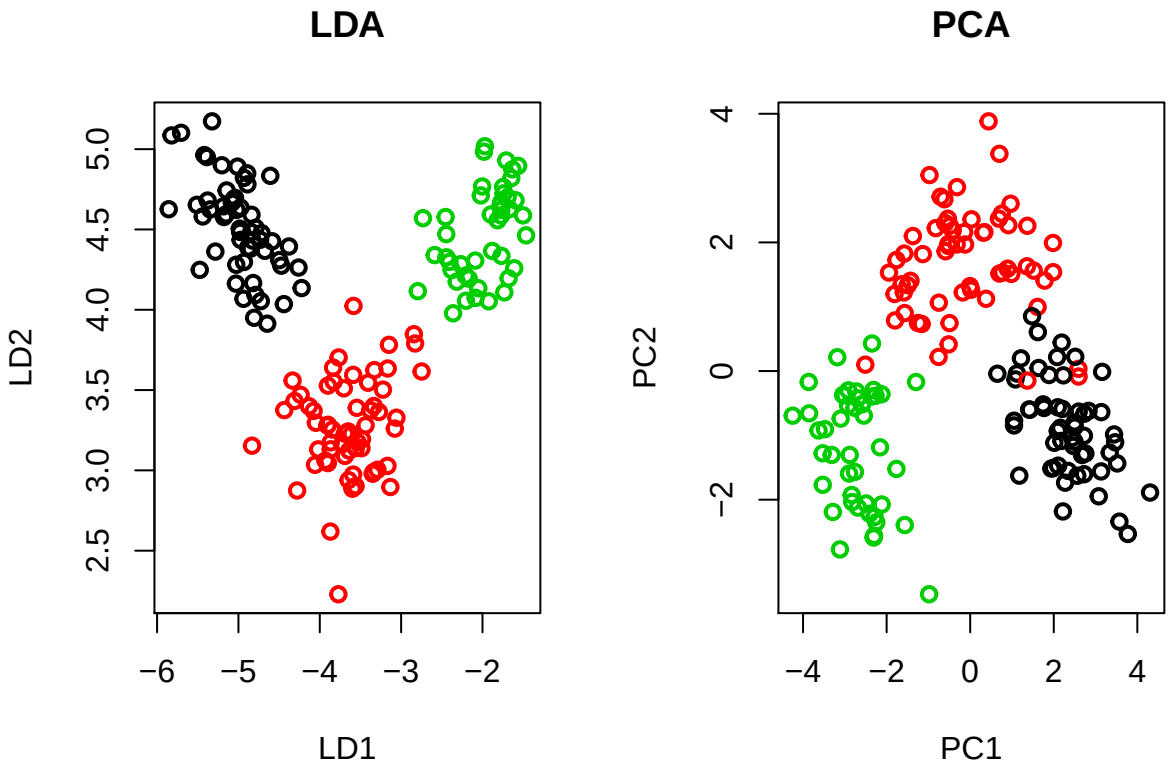

LDA
LD2
LD1
2.5
3.0
3.5
4.0
4.5
5.0
5.5
−5
−4
−3
−2
−1
PCA
PC2
PC1
−2
0
2
4
−4
−2
0
2
4

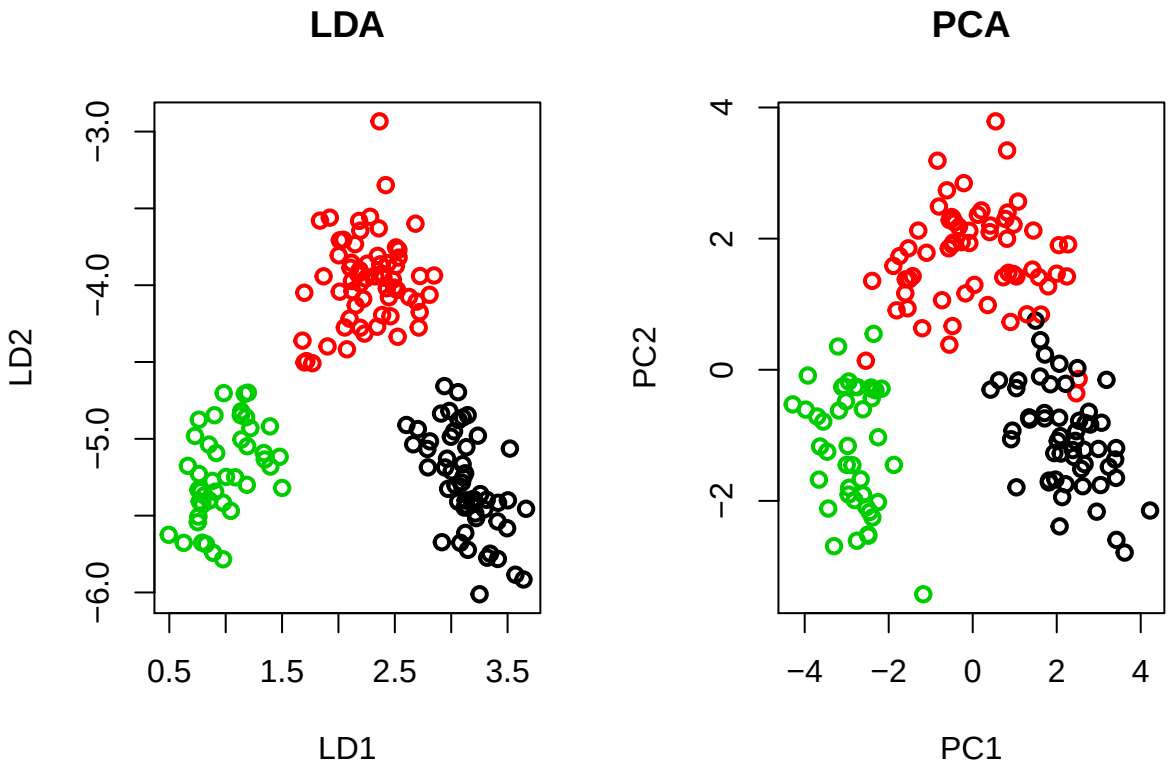

LDA
LD2
LD1
2.5
3.0
3.5
4.0
4.5
−6
−5
−4
−3
−2
PCA
PC2
PC1
−3
−2
−1
0
1
2
3
−4
−2
0
2
4


LDA
LD2
−2.5
−3.5
−4.5
−5.5
1
2
3
4
5
LD1
PCA
PC2
4
2
0
−2
−4
−2
0
2
4
PC1


LDA
LD2
−2.5
−3.5
−4.5
−5.5
2
3
4
5
LD1
PCA
PC2
4
2
0
−2
−4
−2
0
2
PC1


LDA
LD2
−2.5
−3.5
−4.5
−5.5
1
2
3
4
LD1
PCA
PC2
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
PC1


LDA
LD2
LD1
5.0
4.5
4.0
3.5
3.0
2.5
−6
−5
−4
−3
−2
PCA
PC2
PC1
4
2
0
−2
−4
−2
0
2
4



Discuss given results.

Obrázky ukazují, že metoda PCA nebyla schopna bez scaling data dobře separovat. Jelikož PCA metoda funguje na principu maximalizace variance, PCA hledá ty vlastnosti, které toho hodně vysvětlují (mají vysokou varianci). Problém nastává, když v datasetu máme vlastnost, která dominuje a PCA jej preferuje vůči tomu zbytku. Když nastavíme scale = TRUE, tak nastavujeme pro variables 'unit variance', což dovolí PCA lépe data separovat. Přestože PCA není úplně klasifikační metoda, je vhodná pokud máme příliš málo dat (pak není vhodné LDA samo o sobě). V tomto případě ale metoda LDA bez problémů separovala data s úspěšností ~0.989 (1 láhev byla klasifikována špatně).