SADM

Elmehdi TERRAF

12/01/2023

```r
set.seed(0)
```

1. Data

```r
NAm2 <- read.table("NAm2.txt", header = TRUE)

cont <- function (x ){
  if(x %in% c("Canada"))
     cont <- "NorthAmerica"
  else if( x %in% c("Guatemala", "Mexico", "Panama", "CostaRica"))
     cont <-"CentralAmerica"
  else
     cont <-"SouthAmerica"
  return(factor(cont))
}

contID <- sapply(as.character(NAm2[ ,4]), FUN=cont)
```

2. Multinomial regression

```r
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 4.0.5
```

```r
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.5
```

```r
library(class)
```

```
## Warning: package 'class' was built under R version 4.0.5
```

```r
library(naivebayes)
```

```
## Warning: package 'naivebayes' was built under R version 4.0.5
```

```
## naivebayes 0.9.7 loaded
```

```r
NAcont <- cbind(contID = contID , NAm2[ , -(1:8)])
NAcont[ ,1] <- factor(NAcont[ ,1])
#multinom(contID~., data = NAcont)
```

The error message we get ("Error in nnet.default(X, Y, w, mask = mask, size = 0, skip = TRUE, softmax = TRUE, : trop (17133) de pondérations") is telling us that the neural network called by the multinom function uses too many weights. After setting the maximum number of weights to 18000 the problem is solved, because the minimal required number of weights is 17133. However, our algorithm takes a decent time to run, we note that the run-time could be reduced by diminishing the maximum number of iterations allowed.

```r
#multinom(contID~., data = NAcont, MaxNWts = 18000, maxit = 200)
```

We choose not to display the results here since it is not our goal, and this step simply helps us to get started with the `multinom` function for the rest of our study.

b)

```r
#This function will allow to retrieve predicted_values given a number k of pcs
# a validation set valid_set and for a specific classifier
# classifier can be "multinom", "lda" or "bernoullinaivebayes"
get_prediction <- function(classifier, k, valid_set){

  if (is.null(valid_set)){
    pcaNAcont <- prcomp(NAcont[, -1], scale = FALSE)
    pcaDf <- data.frame(pcaNAcont$x)
  }
  else{
    #PCA on training set
    pcaNAcontrain <- prcomp(NAcont[!valid_set, -1], scale = FALSE)
    #Training data set
    pcaDftrain <- data.frame(pcaNAcontrain$x)
    #Building validation data set
    pcaDftest <- data.frame(predict(pcaNAcontrain, newdata = NAcont[valid_set, -1]))
  }

  if (classifier == "multinom"){
    if (is.null(valid_set)){
      model <- multinom(NAcont[,1]~., data = pcaDf[,(1:k)], MaxNWts = 18000,
                        maxit = 200, trace = FALSE)
      predicted_values <- predict(model, data = pcaDf[,(1:k)])
    }
    else{
      if (k == 0){
        model <- multinom(NAcont[!valid_set,1]~1, data = pcaDftrain,
                          MaxNWts = 18000, maxit = 200, trace = FALSE)
      }
      else{
        model <- multinom(NAcont[!valid_set,1]~., data = pcaDftrain[,(1:k)],
                          MaxNWts = 18000, maxit = 200, trace = FALSE)
      }
      predicted_values <- predict(model, newdata = pcaDftest[,(1:k)])
    }
  }

  if(classifier == "lda"){
    if (is.null(valid_set)){ #No validation set
      model <- lda(NAcont[,1]~., data = pcaDf[,(1:k)])
      prediction <- predict(model, data = pcaDf[,(1:k)])
      predicted_values <- prediction$class
    }
    else{
      model <- lda(NAcont[!valid_set,1]~., data = pcaDftrain[,(1:k)])
      prediction <- predict(model, newdata = pcaDftest[,(1:k)])
      predicted_values <- prediction$class
    }
  }

  if (classifier == "bernoullinaivebayes"){
    if (is.null(valid_set)){ #No validation set
      model <- bernoulli_naive_bayes(data.matrix(pcaDf[,(1:k)]), NAcont[,1])
      predicted_values <- predict(model, data = data.matrix(pcaDf[,(1:k)]))
    }
    else{
      model <- bernoulli_naive_bayes(data.matrix(pcaDftrain[,(1:k)]),
                                     NAcont[!valid_set,1])
      predicted_values <- predict(model, newdata = data.matrix(pcaDftest[,(1:k)]))
    }
  }

  return(predicted_values)
}

#Confusion matrix
build_confusion_matrix <- function(classifier, k, valid_set){
    predicted_values <- get_prediction(classifier, k, valid_set)
    if (is.null(valid_set)){
      confusion_matrix <- table(predicted_values, NAcont[,1])
    }
    else{
      confusion_matrix <- table(predicted_values, NAcont[valid_set,1])
    }
    return(confusion_matrix)
}
```

```r
#Confusion matrix for multinom classifier using all pcs on the training set (no validation set)
build_confusion_matrix("multinom",494, NULL)
```

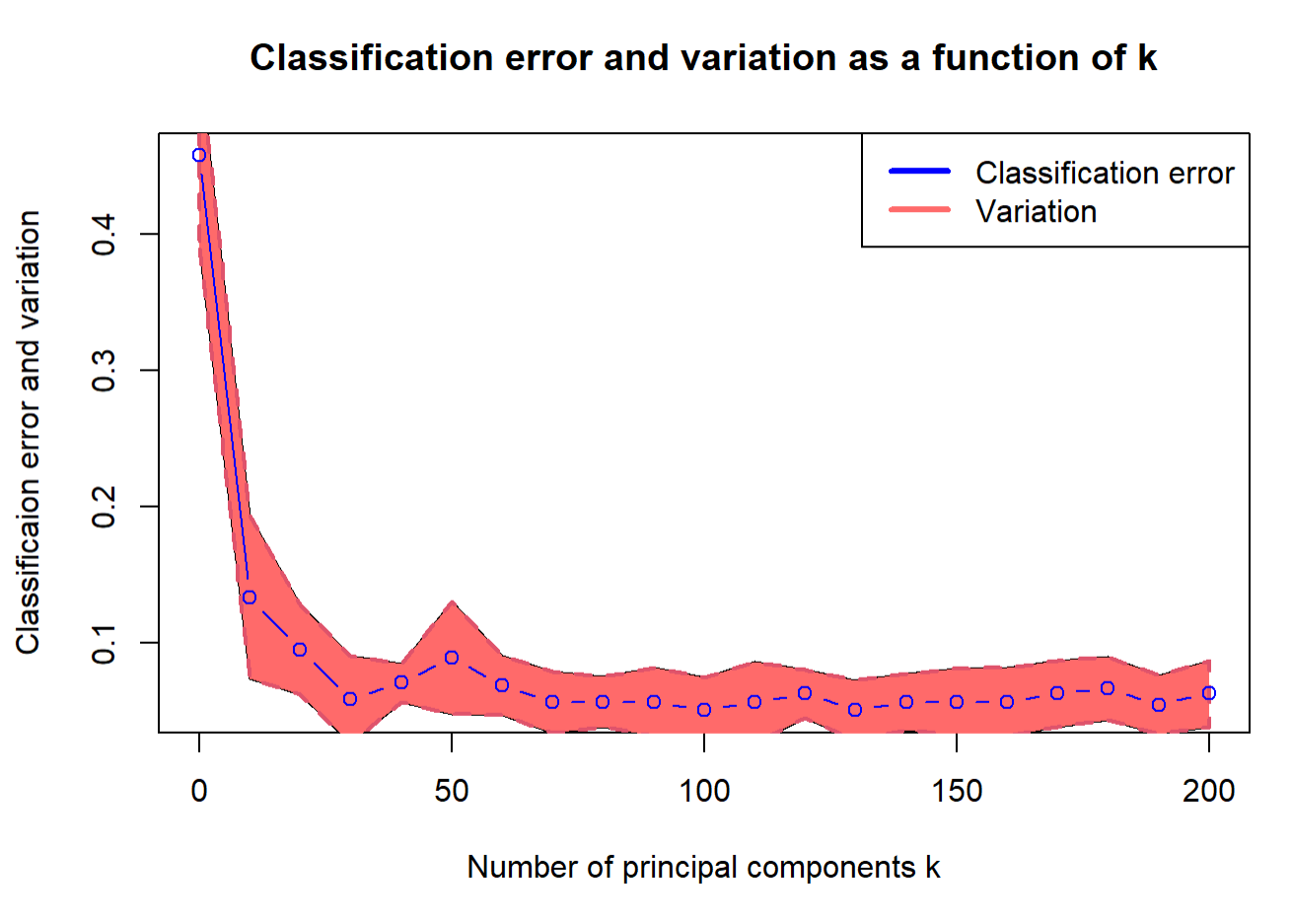
```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica             67              0            0
##   CentralAmerica            0            159            0
##   SouthAmerica              0              0          268
```

We notice thanks to the confusion matrix that no error was made at the level of the prediction, which is normal since in this case the validation set is also the training set and the prediction is made thanks to the use of all principal components.

c)

```r
#Validation sets 1->4 will have 50 elements
#Validation sets 5->10 will have 49 elements
belongs_vector <- sample(c(rep(1:10, each=49), 1:4), 494)

#Gives the classification error by using the confusion matrix
#for a number of pcs k and a certain validation set valid_set
classification_error <- function(classifier, k, valid_set){
  sum <- 0
  confusion_matrix <- build_confusion_matrix(classifier, k, valid_set)
  n <- length(confusion_matrix)
  individuals <- sum(confusion_matrix)
  for (i in (2:(n-1))[-4]){
      sum <- sum + confusion_matrix[i]
  }
  return(sum/individuals)
}

#Gives the error obtained by doing cross validation using k pcs
cross_validation_error <- function(classifier, k){
    errors <- c()
    for (i in 1:10){
      valid <- belongs_vector[1:494] == i
      class_error <- classification_error(classifier, k, valid)
      errors <- c(errors, class_error)
    }
    return(c(mean(errors), sd(errors)))
}

#Allows to display the classification error obtained after cross-validation
# as a function of the number of principal components with the variation associated
display_variation <- function(classifier, sequence){
  classification_errors <- c()
  first_bound <-c()
  second_bound <-c()
  for (k in sequence){
    error <- cross_validation_error(classifier, k)[1]
    variation <- cross_validation_error(classifier, k)[2]
    classification_errors <- c(classification_errors, error)
    #variation 1 mean(errors) + sd(errors)
    first_bound <- c(first_bound, error + variation)
    #variation 2 mean(errors) - sd(errors)
    second_bound <- c(second_bound, error - variation)
  }

  plot(sequence, classification_errors, xlab = "Number of principal components k",
      ylab = "Classificaion error and variation",
      main = "Classification error and variation as a function of k",
      type ="b", col ="blue")
  lines(sequence, first_bound)
  lines(sequence, second_bound)
  polygon(c(sequence, rev(sequence)), c(second_bound, rev(first_bound)),
          col = "indianred1", border = 2, lwd = 2, lty = 2)
  lines(sequence, classification_errors, type = "b", col ="blue")
  legend(x="topright", legend=c("Classification error", "Variation"),
       col=c("blue", "indianred1"), lwd=c(3,3))
}

#Allows to display the classification error obtained after cross-validation
# as a function of the number of principal components
display <- function(classifier, sequence){
  classification_errors <- c()
  for (k in sequence){
    error <- cross_validation_error(classifier, k)[1]
    classification_errors <- c(classification_errors, error)
  }

  plot(sequence, classification_errors, xlab = "Number of principal components k",
      ylab = "Classificaion error", main = "Classification error as a function of k",
      type ="b", col ="blue")
}
```

```r
display_variation("multinom",seq(0,200, by = 10))
```



We choose not to represent for the next values of k. This is because the classification error will only increase, so the rest of the graph is not meaningful except to show that the classification error is increasing, which we already know. We will adopt this approach for the rest of the study. Let's check some error values to ensure this:

```r
cross_validation_error("multinom", 300)[1]
```

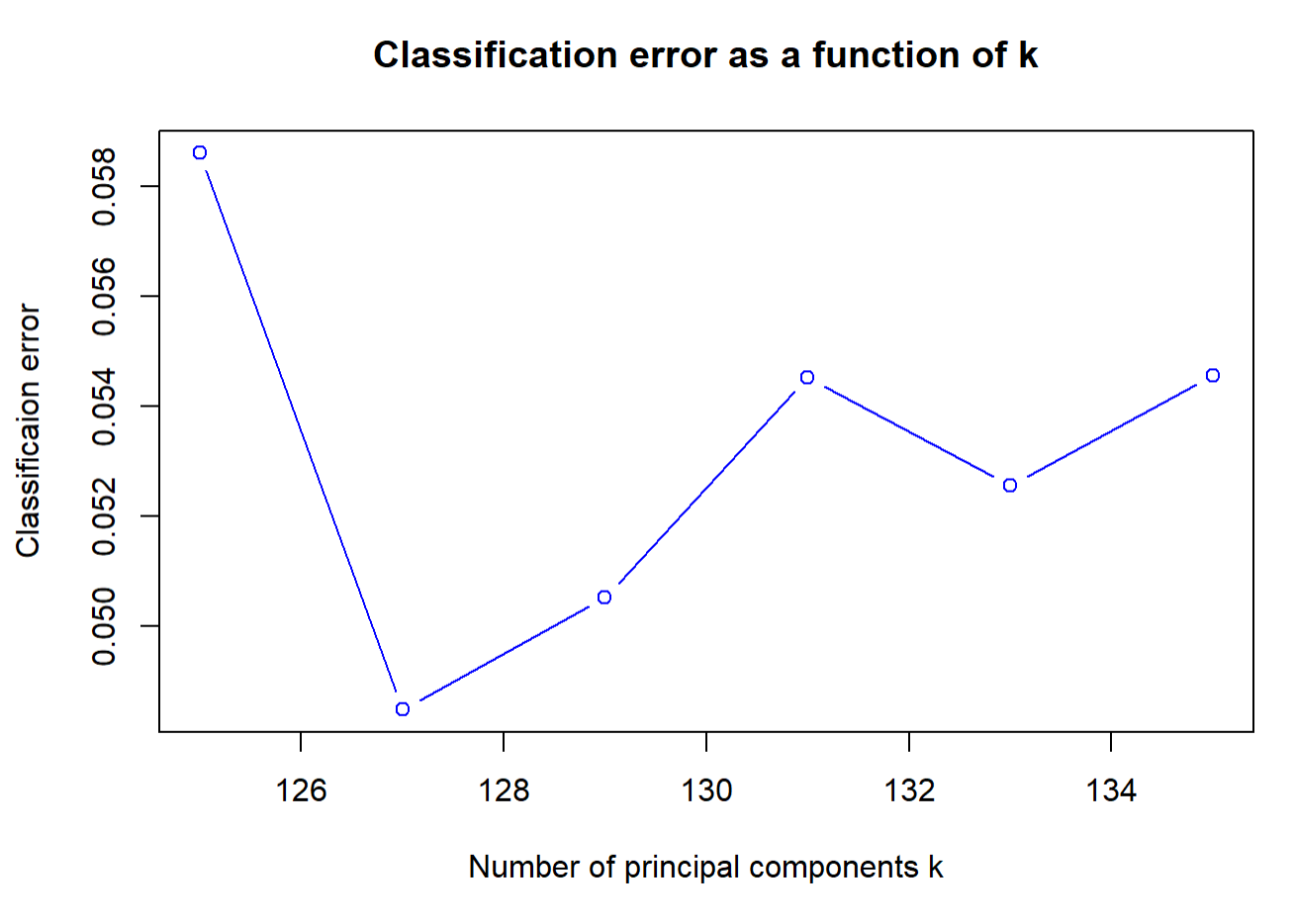
```
## [1] 0.07681633
```

```r
cross_validation_error("multinom", 400)[1]
```

```
## [1] 0.08081633
```

We see that the minimal value of the classification error is reached when $k = 130$. Initially, we therefore choose this number of principal components to minimize the classification error obtained after cross-validation. In order to refine this result, let's observe the variation of the error in an interval close to our initial value. After conducing our analysis and in order to accelerate the computation, whe show here some pertinent values only that will help us choose the best value for the number of principal components :

```r
display("multinom", seq(125,135, by = 2))
```



We then choose $k = 127$ principal components as the best choice to minimize the classification error. Let's observe the confusion matrix obtained by using this number of 127 principal components and by operating the test on the same training set (validation set is NULL in this case) :

d)

```r
#Confusion matrix for multinom classifier using 127 pcs on the training set (no validation set)
build_confusion_matrix("multinom", 127, NULL)
```

```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica             67              0            0
##   CentralAmerica            0            159            0
##   SouthAmerica              0              0          268
```

The result proves that the model has correctly classified the training set. This result is the same as the one obtained above with the maximum number of principal components, however we are now sure thanks to our analysis that this chosen number of principal components is optimal to make the classification on a new database, contrary to the maximum number.

3. Linear discriminant analysis

a)

```r
#Confusion matrix for lda classifier using all pcs on the training set (no validation set)
build_confusion_matrix("lda",493, NULL)
```

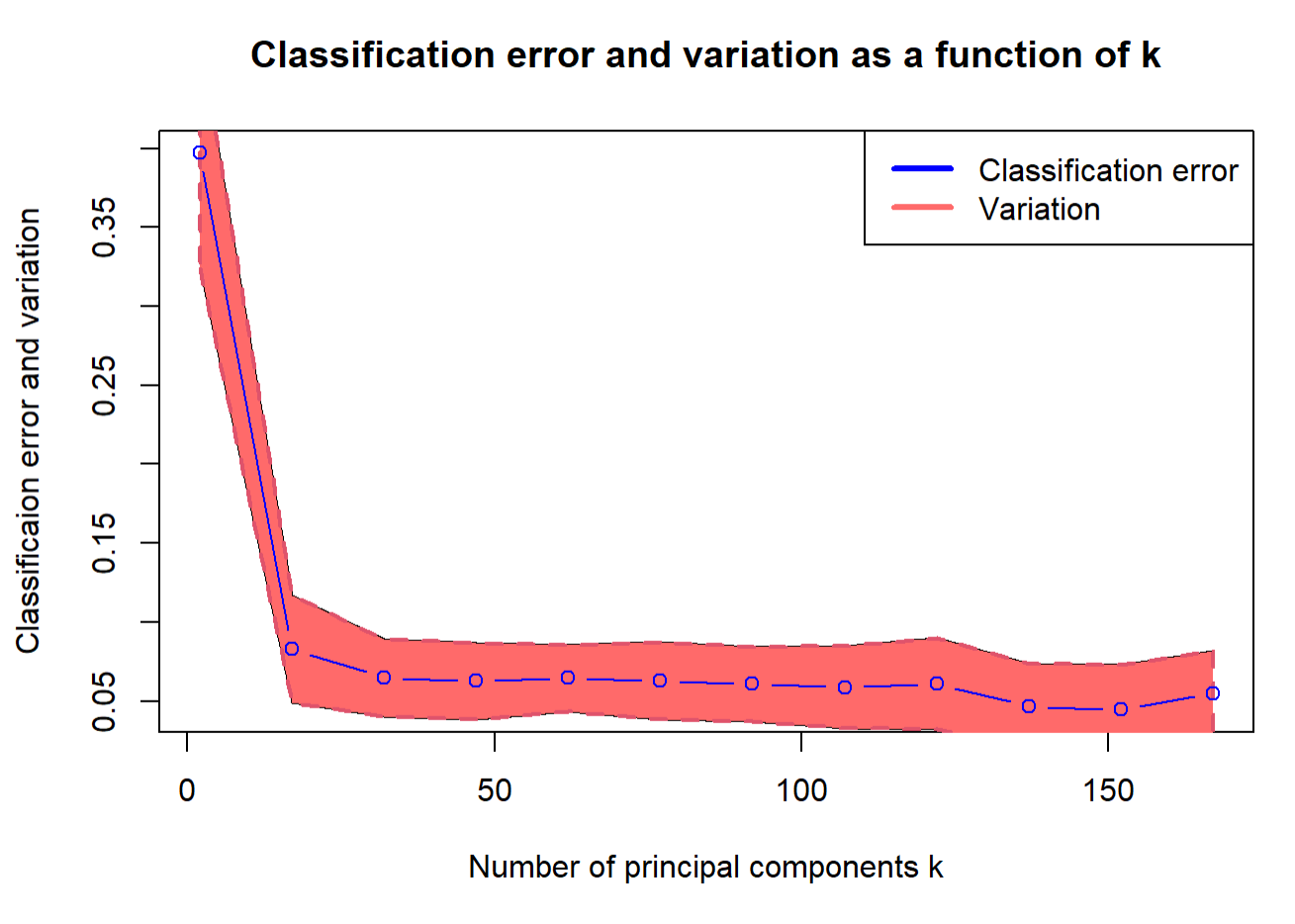
```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica              0              0            3
##   CentralAmerica            0             16            6
##   SouthAmerica             67            143          259
```

Note : Because of the values of the coefficients on the 494 column are too small in the order of 10^{-15}, R consider them as 0. So it could not solve the matrix inverse because the within-class covariance matrix was singular. That gives the error of : variable(s) 494 appear to be constant within groups. So we decide to use 493 principal components instead of 494 as the maximum number of principal components.

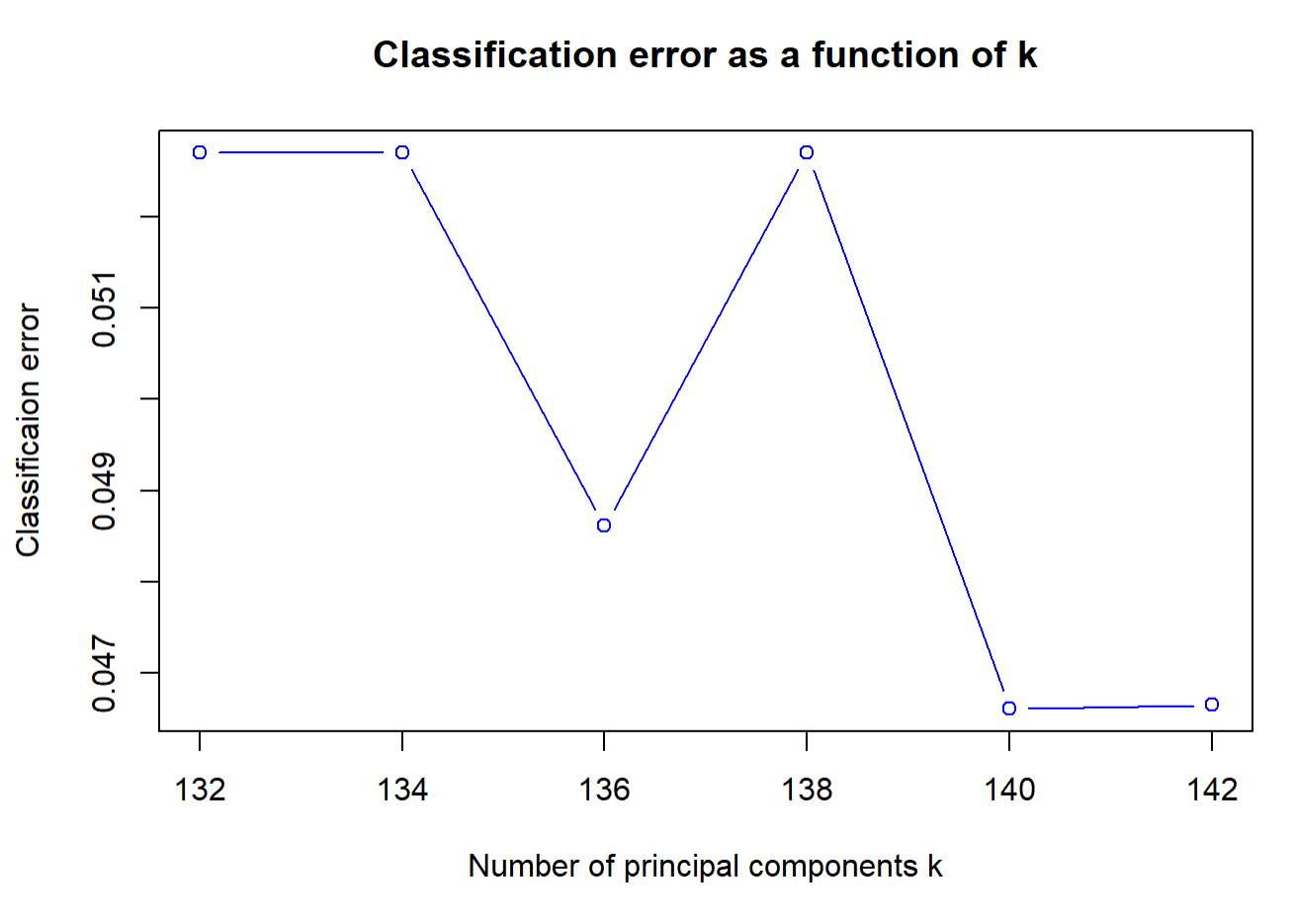
Another remark concerns the appearance of the warning: "variables are collinear". This explains why so many errors were made in the classification using all the principal components compared to what was done using multinomial regression.

```r
#Classification error as a function of principal components for lda classifier
display_variation("lda",seq(2,180, by = 15))
```



Here again we only show the relevant values for our study. It seems that the minimal value for the error is reached for $k = 137$. As we did before, let's refine this value by considering :

```r
display("lda",seq(132,142, by = 2))
```



The optimal number of principal components in order to minimize the classification error for `lda` classifier is therefore $k = 140$.

```r
#Confusion matrix for lda classifier using 140 pcs on the training set (no validation set)
build_confusion_matrix("lda",140, NULL)
```

```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica             67              0            0
##   CentralAmerica            0            157            0
##   SouthAmerica              0              2          268
```

We notice that the classification made here on the training set is quite satisfactory and is even better than the one made with the maximum number of principal components. Which doesn't seem logical, since we are working on the training set in this case and more principal components should give a better result. This proves the malfunction of the LDA classifier for our data.

b) LDA classification seems to produce more errors than classification done with multinomial regression. This is due to the fact that the variables are correlated, which decreases the performance of the LDA.

4. Naive Bayes:

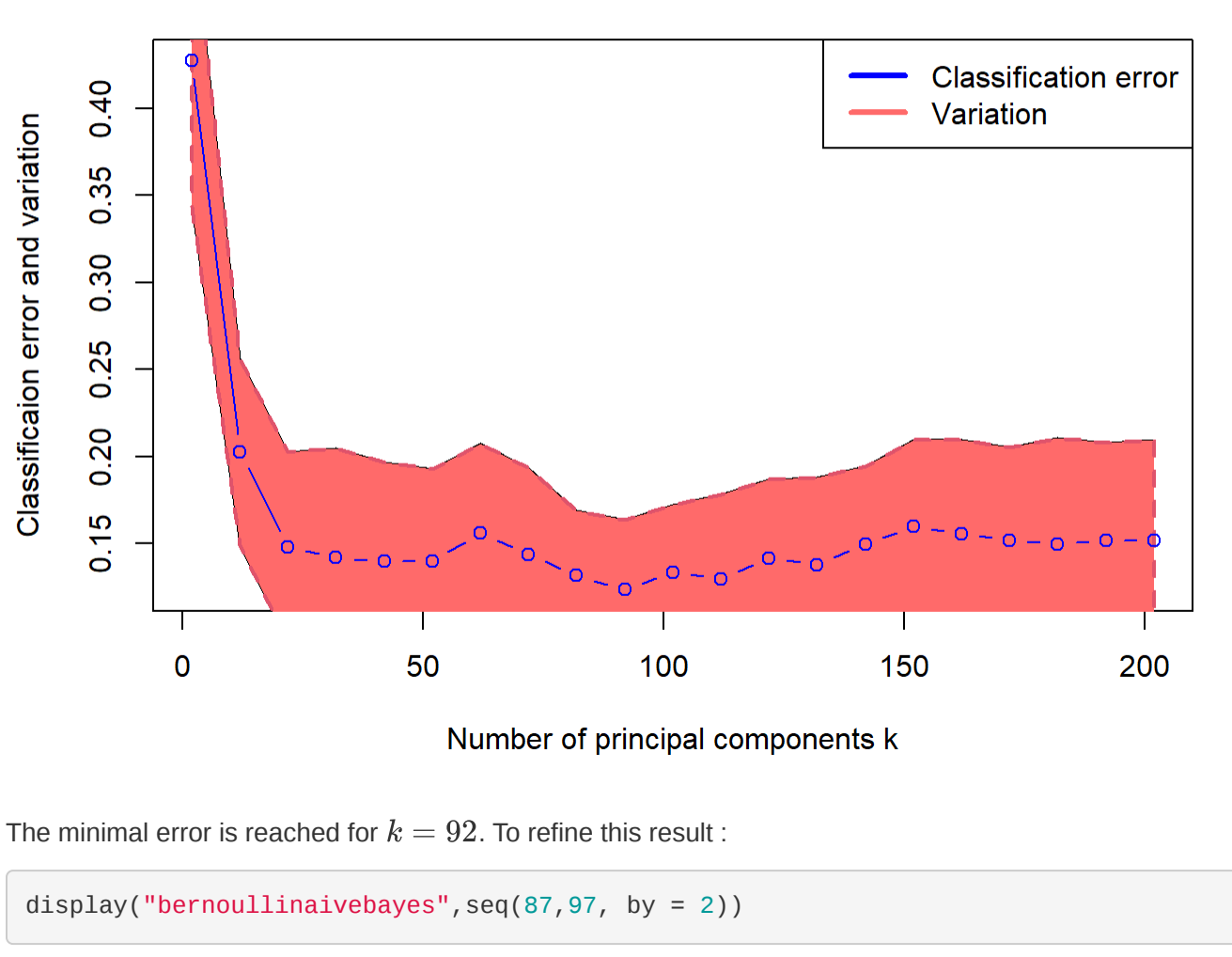
a)

```r
#Confusion matrix for naivebayes classifier using all pcs on the training set (no validation set)
build_confusion_matrix("bernoullinaivebayes",494, NULL)
```

```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica             67              4            5
##   CentralAmerica            0            155           13
##   SouthAmerica              0              0          250
```

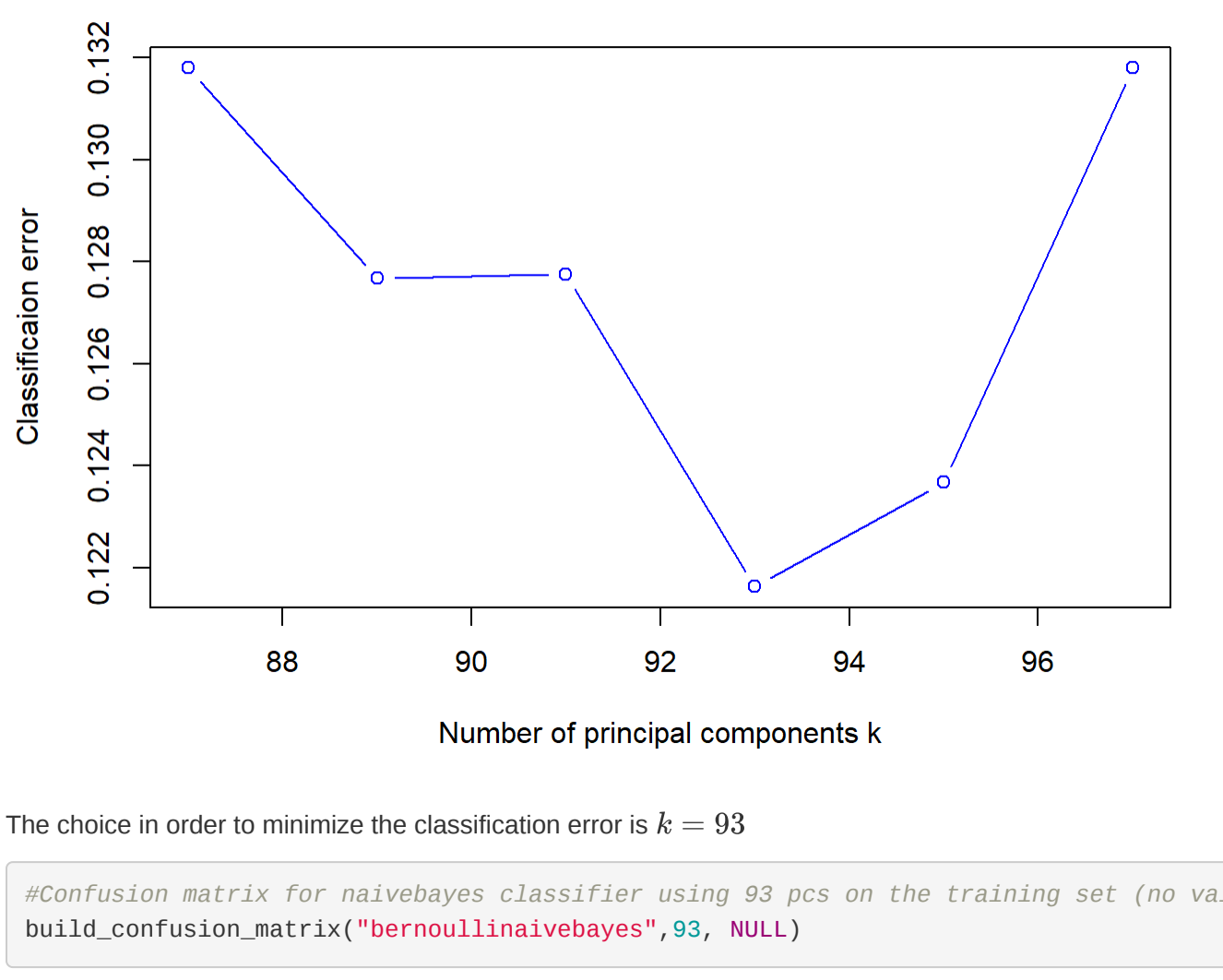
Some errors are made in the classification from the training set and using all the principal components available.

```r
#Classification error as a function of principal components for naivebayes classifier
display_variation("bernoullinaivebayes",seq(2,202, by = 10))
```



The minimal error is reached for $k = 92$. To refine this result :

```r
display("bernoullinaivebayes",seq(87,97, by = 2))
```



The choice in order to minimize the classification error is $k = 93$

```r
#Confusion matrix for naivebayes classifier using 93 pcs on the training set (no validation set)
build_confusion_matrix("bernoullinaivebayes",93, NULL)
```

```
##
## predicted_values NorthAmerica CentralAmerica SouthAmerica
##   NorthAmerica             67             11           14
##   CentralAmerica            0            142           32
##   SouthAmerica              0              6          222
```

Some errors are made at the level of the classification and the result is worse than what we obtained with all the main components. This makes sense since we are working on the training set in this specific case.

b)

We notice globally that the multinomial classifier performs better than the others. This is explained by the fact that the variables are correlated, as it was raised during the execution of the LDA. This decreases the performance of the latter and irregularities have been perceived up to date (in particular the training error which increases with the increase in the number of principal components). With regard to the naive bayes approach, it gives the greatest errors of classification and this is due to the fact that the main hypothesis of this model of classification is to consider the variables independent, which is not our case. And which therefore distorts the results of this classifier. Let us compare for example the minimal errors obtained by the multinomial classifier and the naive bayes one:

```r
#Minimum possible error value
cross_validation_error("multinom", 127)[1]
```

```
## [1] 0.0484898
```

```r
cross_validation_error("bernoullinaivebayes", 93)[1]
```

```
## [1] 0.1216327
```

We therefore punctuate our analysis by choosing the **multinomial regression** classifier as being the most suitable for the analysis of our data. This classifier gives encouraging results with regard to training data and promises satisfactory results on a foreign database by choosing 127 principal components.

Note: the execution and generation time of this file is extremely long. It is for this reason that we chose the values of principal components to display in each graph and thus save time, after having of course conclude that the values we left out were not relevant.