TP2 - Statistical analysis and document mining

Youssef BENHACHEM, Mouad BOUCHNAF, Amine EL BOUZID

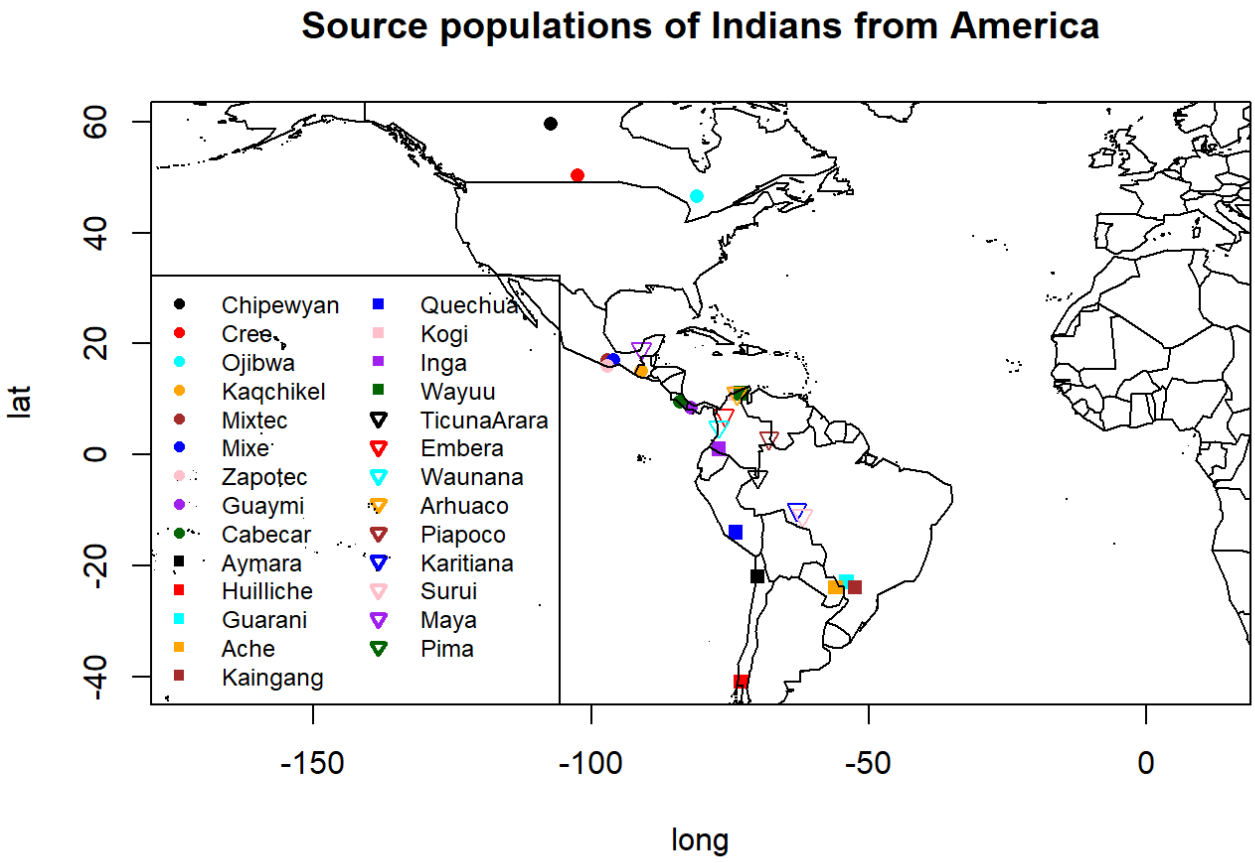
28/02/2022

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

1. Data

```
#Retrieves the names of different populations in a unique way
names = unique(NAm2$Pop)
npop = length(names)

#Retrieves the coordinates of each type of population
coord = unique (NAm2[ ,c("Pop","long","lat" )])

#Configure the colors to use (27 populations)
colPalette =rep(c("black","red","cyan","orange","brown","blue","pink",
                  "purple","darkgreen") ,3)
#Configure the type of points to use
#each type of points is used 9 times with the 9 colors above
pch = rep(c(16 ,15 ,25) , each=9)

#The plot function that uses the longitude and latitude coordinates
#for each population
plot(coord[,c("long","lat")] , pch = pch , col = colPalette , asp =1,
     main = "Source populations of Indians from America")
legend("bottomleft",legend =names , col = colPalette , lty = -1 ,
pch = pch , cex =0.75 , ncol =2 , lwd =2)

#The plot is made on the world's map
library(maps) ; map("world",add =T )
```

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



2. Regression

```
NAaux <- NAm2[,-c(1:7)]
reg <- lm(long~., data=NAaux)
#Allow to show the beginning of the summary instead of 5000+ rows
regOut <- capture.output(summary(reg))
for (i in 2:22){
  print(regOut[i], quote = FALSE)
}
```

```
## [1] Call:
## [1] lm(formula = long ~ ., data = NAaux)
## [1]
## [1] Residuals:
## [1] ALL 494 residuals are 0: no residual degrees of freedom!
## [1]
## [1] Coefficients: (5216 not defined because of singularities)
## [1]                          Estimate Std. Error t value Pr(>|t|)
## [1] (Intercept)            1250.3745         NA      NA       NA
## [1] L1.125                 -171.2557         NA      NA       NA
## [1] L1.130                   44.7543         NA      NA       NA
## [1] L1.135                  -54.5783         NA      NA       NA
## [1] L1.140                 -264.3330         NA      NA       NA
## [1] L1.142                 -792.4778         NA      NA       NA
## [1] L1.145                   65.6649         NA      NA       NA
## [1] L1.150                   51.0931         NA      NA       NA
```

```
## [1] L1.150.940397350993      25.7019         NA      NA       NA
## [1] L1.152                 -249.8699         NA      NA       NA
## [1] L1.155                   25.2062         NA      NA       NA
## [1] L1.160                   67.5255         NA      NA       NA
## [1] L1.165                        NA         NA      NA       NA
```

The regression performed returns missing values, which shows that the model is not relevant and does not allow the desired prediction to be made.

We notice also that the Estimate coefficients of some variables(L1.165, L29.159, L26.186,…) has a value of NA. This is indicates that the variables in question are linearly related to the other variables. then there is no unique solution to the regression without dropping some variables, and that's what we are going to do with the PCA method.

\(\textbf{Why we cannot do a multiple linear regression on this model :}\)

Since p > n then the column vectors of the design matrix X of dimension n×p are not independent, we can show it mathematically by posing a lambda vector of dimension p×1 and we put the product of the matrices at 0 and then showing that there is a dependence between variables. But we can see it as we already said from the results of \(\texttt{summary()}\). In fact since there are coefficients with NA value then there is the dependence between the variables so the column vectors. So the set of column vectors is not a basis and so the matrix \(\mathbf{X}^\intercal X\) is not invertible . Thus there isn't a unique least squares coefficient estimate, then the variance is infinite so we cannot used this method at all.

3. PCA

\(\textbf{a)}\) The principal component analysis is a method of dimension reduction whose goal is to reduce the dimension (i.e number of variables) of a large data set to have a smaller and more easily manipulated set. This reduction is made so that no loss of information is to be deplored despite the reduction made.

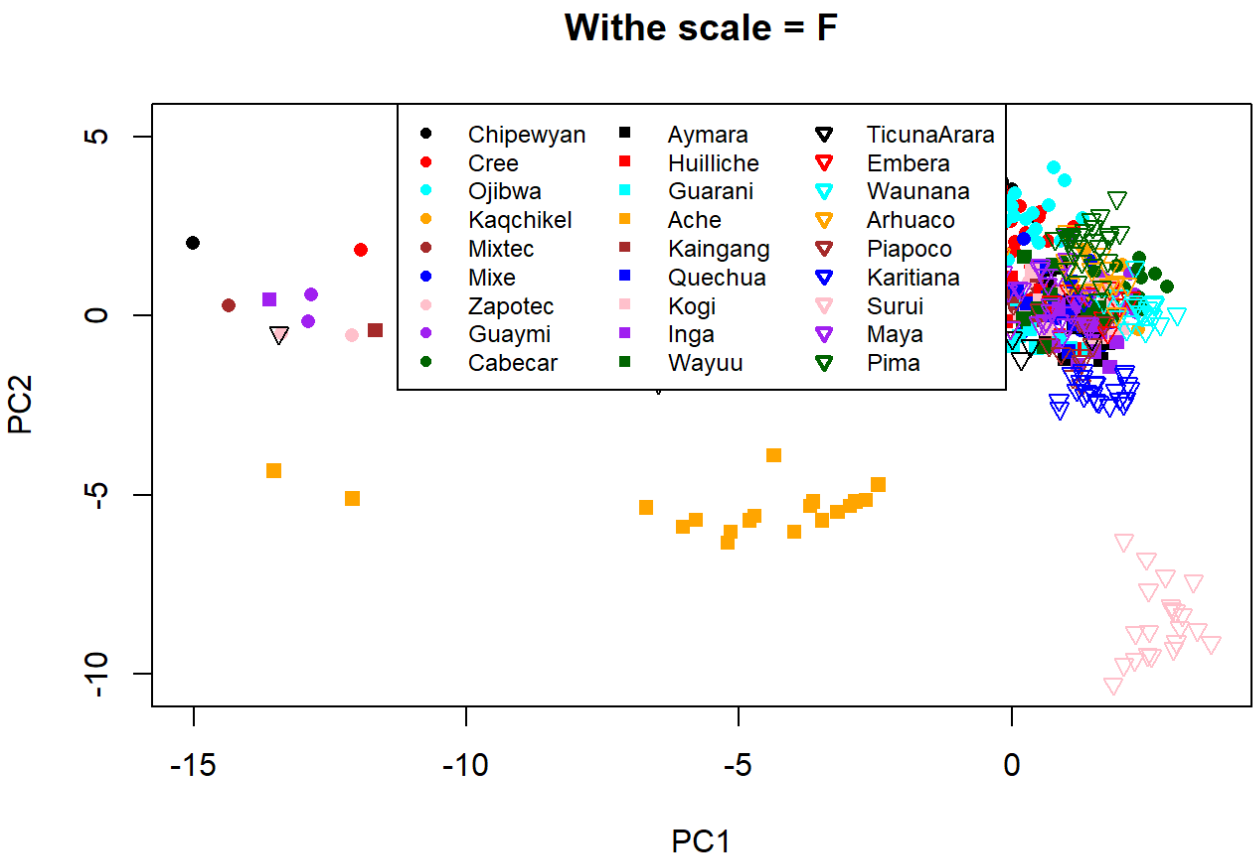
\(\textbf{b)}\)

```
NAgen <- NAm2[,-c(1:8)]
pcaNAm2 <- prcomp(NAgen)
```

There is no need to use the argument \(\texttt{scale}\) of the function \(\texttt{prcomp}\), since the values of the genetic codes are either 0 or 1.

\(\textbf{c)}\)

```
#scale = FALSE
caxes =c(1,2)
plot(pcaNAm2$x[,caxes],col ="white", main = "Withe scale = F")
for (i in 1:npop){
  print(names[i])
  lines(pcaNAm2$x[which(NAm2[ ,3]==names[i]),caxes],type ="p",
        col = colPalette[i], pch = pch [ i ])
  legend ("top",legend =names , col = colPalette ,
          lty = -1 , pch = pch , cex =0.75 , ncol =3 , lwd =2)
}
```

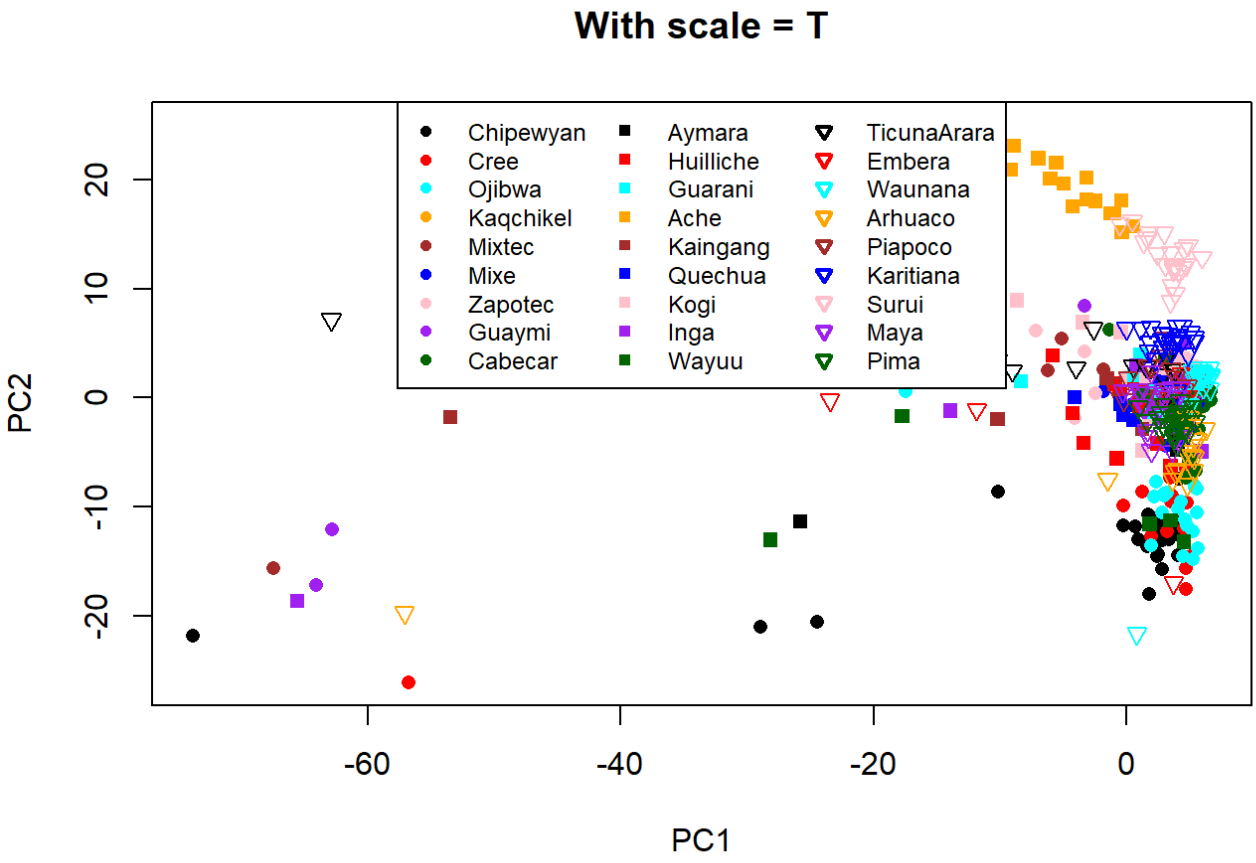



```
## [1] "Chipewyan"
## [1] "Cree"
## [1] "Ojibwa"
## [1] "Kaqchikel"
## [1] "Mixtec"
## [1] "Mixe"
## [1] "Zapotec"
## [1] "Guaymi"
## [1] "Cabecar"
## [1] "Aymara"
## [1] "Huilliche"
## [1] "Guarani"
## [1] "Ache"
## [1] "Kaingang"
## [1] "Quechua"
## [1] "Kogi"
## [1] "Inga"
## [1] "Wayuu"
## [1] "TicunaArara"
## [1] "Embera"
## [1] "Waunana"
## [1] "Arhuaco"
## [1] "Piapoco"
## [1] "Karitiana"
## [1] "Surui"
## [1] "Maya"
## [1] "Pima"
```

The majority of the populations are located at the top right corner of the graph, which means they all do have strong values of PC1 and PC2.

Thus, the majority of them are mixed, which means that we can't tell the difference between them, this is according to the variability that explain the two axes which is not sufficient. We conclude then that the control and treatment are similar, there is no treatment effect.

```
#scale = TRUE
pcaNAm2_true <- prcomp(NAgen, scale = TRUE)
plot(pcaNAm2_true$x[,caxes],col ="white", main = "With scale = T")
for (i in 1:npop){
  lines(pcaNAm2_true$x[which(NAm2[ ,3]==names[i]),caxes],type ="p",
        col = colPalette[i], pch = pch [ i ])
  legend ("top",legend =names , col = colPalette ,
          lty = -1 , pch = pch , cex =0.75 , ncol =3 , lwd =2)
}
```



After setting the scale argument to TRUE, we notice that the ranges of PC1 and PC2 become wider : from [-15,5] to [-60,5] for PC1, and from [-10,5] to [-20,20] for PC2.

It is completely reasonable that the principal components change since the covariance matrix is affected due to the scaling factor. In this example, since all the genetic code values are between 0 and 1, the scaling is not necessary and its implementation will cause us a loss of information.

\(\textbf{c)}\)

```
#cumulative variance for the two first components
summary(pcaNAm2)$importance[3,2]
```

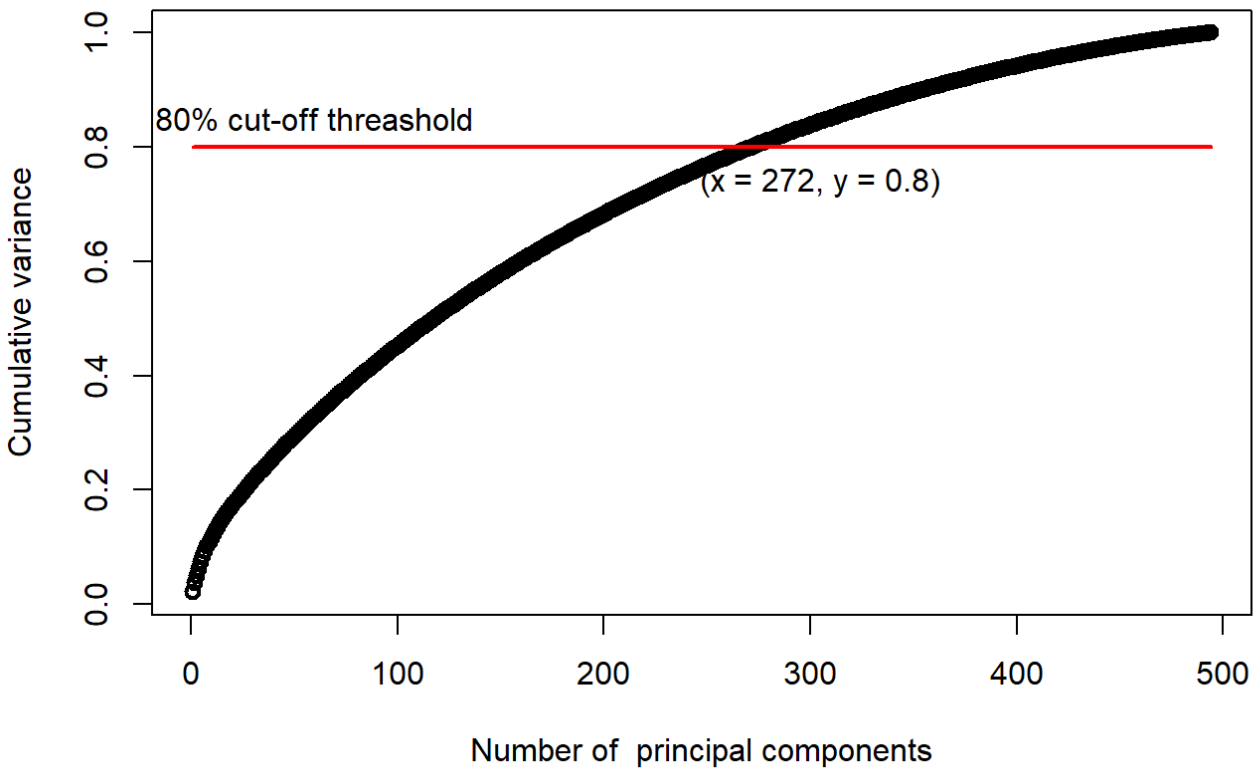
```
## [1] 0.03568
```

The proportion of variance captured by the two first principal components is 0.03568, which is 3.568%.

We note that this percentage of variance is too low to interpret our data with only 2 principal components, which confirms what we just said in the previous question about the map.

To know which principal components we're going to keep, we'll plot the cumulative variance in function of the number of components.

```
plot(1:494, summary(pcaNAm2)$importance[3,], lwd = 2,
     xlab = "Number of  principal components", ylab= "Cumulative variance")
lines(1:494, rep(0.80,494), col = "red", lwd =2)
text(60,0.85,"80% cut-off threashold")
text(305,0.74, "(x = 272, y = 0.8)")
```



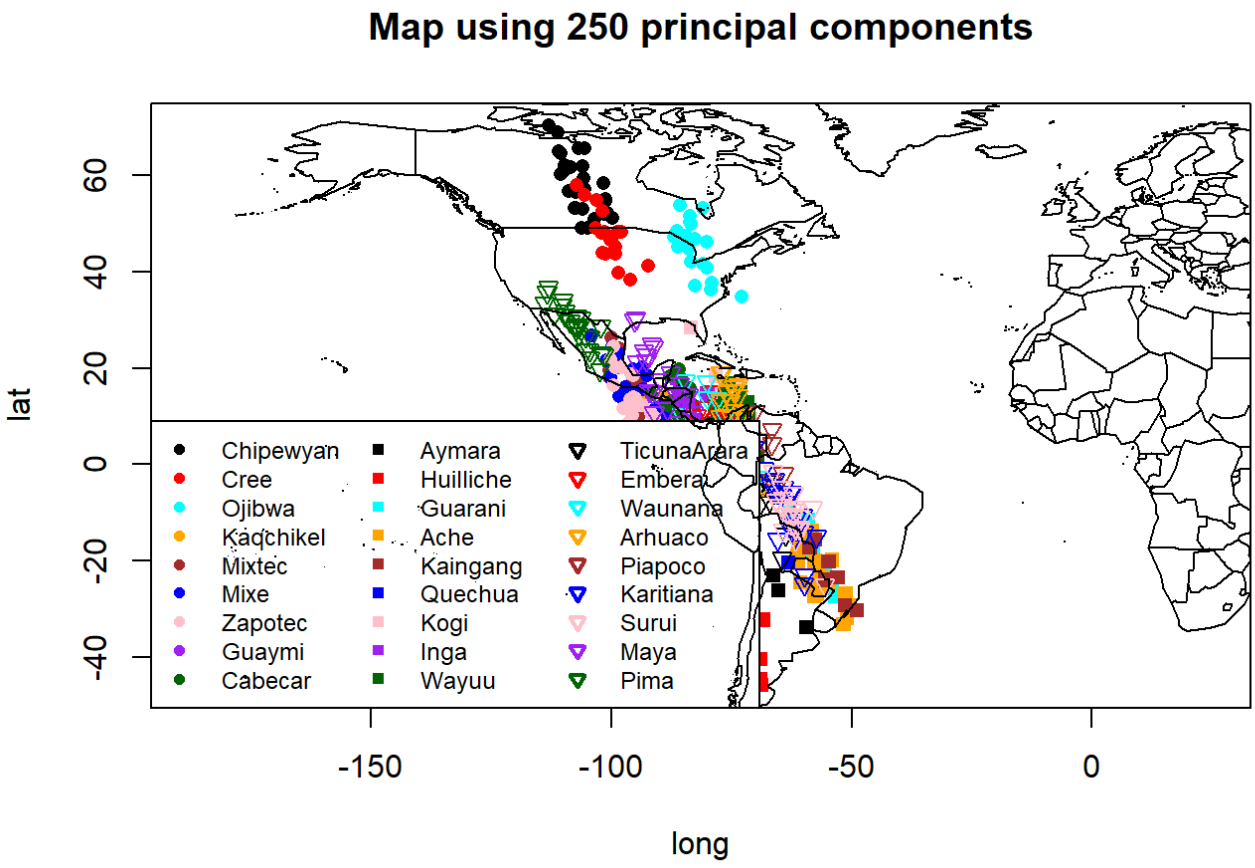
To represent the genetic markers, we would use 272 principal components, since it will guarantee us a cumulative variance of 80%. The function plotted above is like a square root function : "its derivative decreases with time", which means the more cumulative variance we want the more principal components we will have to add. This is the reason why we stopped at 80%.

4. PCR

\(\textbf{a)}\)

```
#250 principal components
pcaDf <- data.frame(pcaNAm2$x)
```

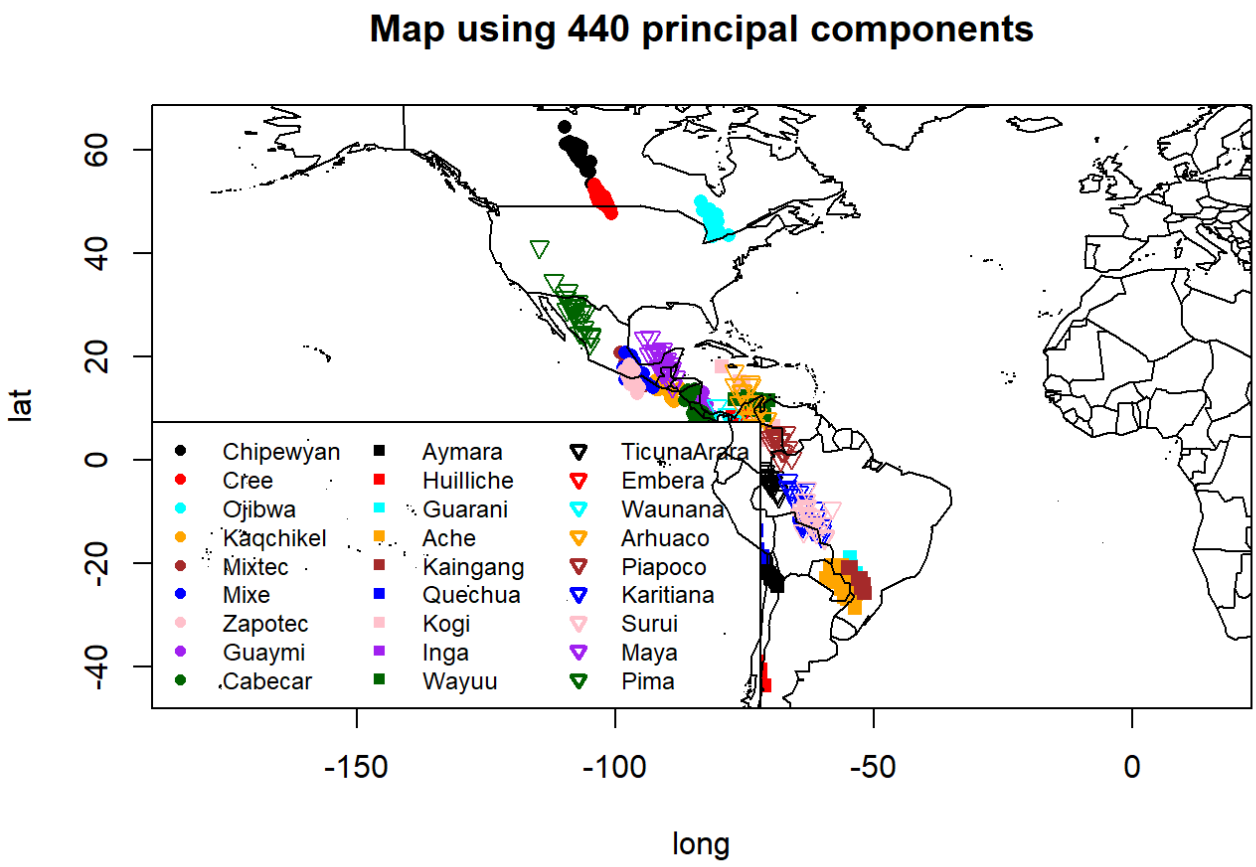
```
lmlat <- lm(NAm2[,7]~., data=pcaDf[,1:250])
lmlong <- lm(NAm2[,8]~., data=pcaDf[,1:250])
plot(lmlong$fitted.values, lmlat$fitted.values, col ="white", xlab="long",
     ylab="lat" ,asp =1, main = "Map using 250 principal components")
for ( i in 1: npop ){
  lines(lmlong$fitted.values[which(NAm2[ ,3]==names[i])] ,
        lmlat$fitted.values[which(NAm2[ ,3]==names[i])] ,
        type ="p", col = colPalette[i], pch = pch[i])
}
legend ("bottomleft",legend =names , col = colPalette , lty = -1 ,
        pch = pch , cex =.75 , ncol =3 , lwd =2)
map ("world",add = T)
```



We notice that in this map, the geographical origins are rather well represented and well correlated between them, which can at first sight be able to predict the geographical origins of a population outside our database. But this representation is actually quite optimistic, since the prediction is made on the same set where the training was done. It is therefore normal to have a good result for a large number of predictors since the training error only decreases according to the complexity of the model. Note this by taking 440 predictors:

```
#440 principal components
pcaDf <- data.frame(pcaNAm2$x)
lmlat <- lm(NAm2[,7]~., data=pcaDf[,1:440])
lmlong <- lm(NAm2[,8]~., data=pcaDf[,1:440])
plot(lmlong$fitted.values, lmlat$fitted.values, col ="white", xlab="long",
     ylab="lat" ,asp =1, main = "Map using 440 principal components")
for ( i in 1: npop ){
  lines(lmlong$fitted.values[which(NAm2[ ,3]==names[i])] ,
```

```
        lmlat$fitted.values[which(NAm2[ ,3]==names[i])] ,
        type ="p", col = colPalette[i], pch = pch[i])
}
legend ("bottomleft",legend =names , col = colPalette , lty = -1 ,
        pch = pch , cex =.75 , ncol =3 , lwd =2)
map ("world",add = T)
```



The result is even better, and this confirms what was said above. However, \(\textbf{beware of the prediction error!}\). We cannot calculate it at the moment, but taking a large number of principal components will surely expose us to the risk of \(\texttt{overfitting}\) and the model with 250 principal components will no longer be effective on a new dataset.The study of the prediction error will be the subject of the 5th part, and we can then decide on the optimality of the model with 250 predictors..

\(\textbf{b)}\)

```
library(fields)
```

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

```
## Loading required package: spam
```

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

```
## Spam version 2.8-0 (2022-01-05) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
```

```
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: viridis
```

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

```
## Loading required package: viridisLite
```

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

```
##
## Attaching package: 'viridis'
```

```
## The following object is masked from 'package:maps':
##
##     unemp
```

```
##
## Try help(fields) to get started.
```

```
predicted_set <- cbind(lmlong$fitted.values, lmlat$fitted.values)
actual_set <- cbind(NAm2[,8], NAm2[,7])
error <- mean(rdist.earth(predicted_set, actual_set, miles = FALSE))
error
```

```
## [1] 3650.206
```

We notice that the mean error is quite big than what we are used to see. Actually it is expected because we have used great circle distance which includes 2 errors, the error of the latitude and the one of the longitude. So if we calculate the error of the longitude alone for example we will find an error of the order of 30. In short if we have an error of 30 in longitude and of the same order or less in latitude it is normal to have an error of 3000 for a distance in Km.

5. PCR and Cross-Validation

\(\textbf{a)}\) The cross-validation technique consists of dividing the dataset into K subsets. At each iteration, the training is performed on K-1 sets and the tests are performed on a the remaining set. This

makes it possible to perform the tests on several sets and to take the average error of prediction, and this would then save us from choosing a validation set. A choice that is both complicated and crucial.

```
#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)
```

\(\textbf{b)}\)

```
#1.Creation of an empty database
m <- matrix( nrow = 494, ncol = 2)
predictedCoord <- data.frame(m)
names(predictedCoord) <- c("longitude", "latitude")
#Shows the head
head(predictedCoord)
```

```
##   longitude latitude
## 1        NA       NA
## 2        NA       NA
## 3        NA       NA
## 4        NA       NA
## 5        NA       NA
## 6        NA       NA
```

```
#2 and 3.Latitude and longitude training

#This function gives the indices of rows related to a certain validation set
build_validation_indices <- function(number_valid_set){
  indices <- c()
  for (i in 1:494){
    if(belongs_vector[i] == number_valid_set){
      indices <- c(indices, c(i))
    }
  }
  return(indices)
}

#This function allows the training on the training set
#and store predictions made on the validation set in predictedCoord
train_lat_long <- function(number_valid_set, naxes){
  valid <- belongs_vector[1:494] == number_valid_set
  lmlatcv <- lm(NAm2[!valid,7]~., data=pcaDf[!valid,1:naxes])
  lmlongcv <- lm(NAm2[!valid,8]~., data=pcaDf[!valid,1:naxes])
  longpredict <- predict(lmlongcv, newdata=pcaDf[valid,1:naxes])
  latpredict <- predict(lmlatcv, newdata=pcaDf[valid,1:naxes])
  for (i in 1:length(longpredict)){
    values <- c(longpredict[i], latpredict[i])
    predictedCoord[build_validation_indices(number_valid_set)[i],] <<- values
  }
}

#Allows to calculate the training error of the model using a
#certain validation set
```

```
#The training error would be the mean of the errors of both predictions for
#lat and long
training_error <- function(number_valid_set, naxes){
  valid <- belongs_vector[1:494] == number_valid_set
  lmlatcv <- lm(NAm2[!valid,7]~., data=pcaDf[!valid,1:naxes])
  lmlongcv <- lm(NAm2[!valid,8]~., data=pcaDf[!valid,1:naxes])
  longpredict <- predict(lmlongcv, pcaDf[!valid,1:naxes])
  latpredict <- predict(lmlatcv, pcaDf[!valid,1:naxes])
  error_long <- mean((NAm2[!valid,8]-longpredict)^2)
  error_lat <- mean((NAm2[!valid,7]-latpredict)^2)
  train_error <- mean(error_long, error_lat)
  return(train_error)
}

#In our case we have number_valid_set = 1 and naxes = 4
train_lat_long(1,4)

#At this point, predictedCoord contains predicted values
#from points that are in validation set 1
na.omit(predictedCoord)
```

```
##      longitude   latitude
## 2   -105.07764  43.455004
## 15  -104.73710  42.786763
## 18  -103.91203  42.213882
## 21  -106.09237  44.239352
## 28  -106.53506  45.751318
## 30   -94.28348  28.239742
## 39   -95.61748  29.895810
## 42   -96.56222  31.458040
## 73   -88.26441  18.809390
## 81   -81.47992  12.814287
## 82   -86.18567  15.996837
## 90   -82.57375  11.099179
## 98   -84.13131  13.080161
## 100  -92.83091  25.797764
## 106  -83.39945  12.255398
## 114  -77.03612   2.651637
## 142  -70.48172  -6.244792
## 151  -78.75899   9.709180
## 168  -76.20865   1.845610
## 175  -75.67425   1.540730
## 183  -85.89437  15.592098
## 189  -79.49582   6.316551
## 194  -83.71524  12.278854
## 210  -88.73033  20.011013
## 219  -77.59141   3.341723
## 220  -78.83799   4.715991
## 221  -83.42201  11.829164
## 245  -86.95096  16.960695
## 249  -80.37701   8.495884
## 257  -83.58157  11.943342
## 259  -81.58731   8.758669
## 265  -80.42736   8.055412
```

```
## 285  -69.39658  -7.808866
## 287  -75.50042   1.175531
## 296  -81.50519  12.852363
## 315  -79.60194   8.667366
## 319  -90.60882  22.862742
## 331  -81.36898   9.483983
## 336  -76.56684   1.567617
## 343  -80.92991   8.596171
## 351  -96.74015  31.811528
## 387  -76.86779   2.701807
## 397  -85.25985  14.286896
## 411  -68.04199 -10.604808
## 415  -67.55493 -11.881151
## 423  -68.45513  -9.897094
## 458  -84.87359  14.402010
## 460  -82.47880  10.500659
## 476  -98.30483  33.366568
## 488  -93.90067  27.212884
```

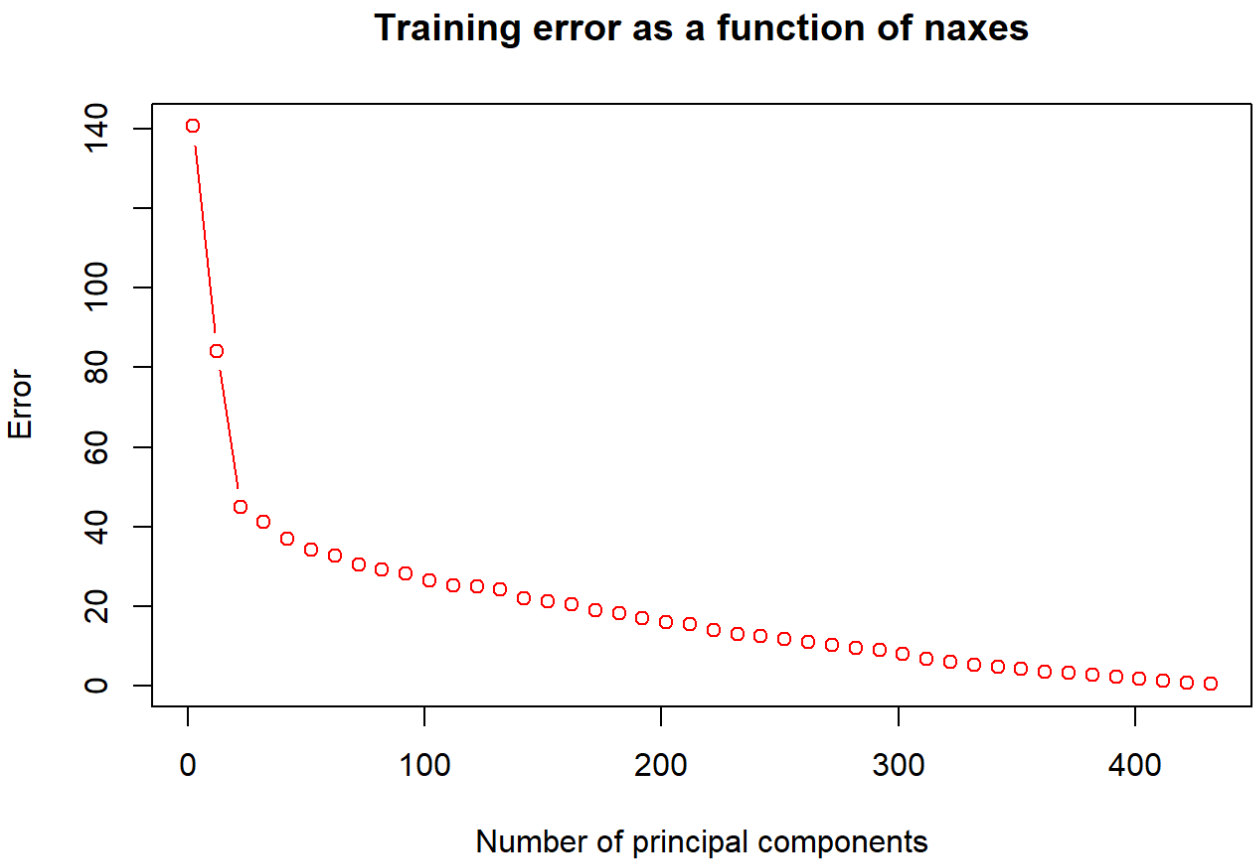
```
#4. for all the validation sets

#Prediction error by using rdist.earth
rd_predict_error <- function(naxes){
  for (k in 2:10){
    train_lat_long(k,naxes)

  }
  #Now predictedCoord is full
  prediction_error <- mean(rdist.earth(predictedCoord, actual_set,
                                       miles = FALSE))
  return(prediction_error)
}

#Prediction error in our case (naxes = 4)
rd_predict_error(4)
```

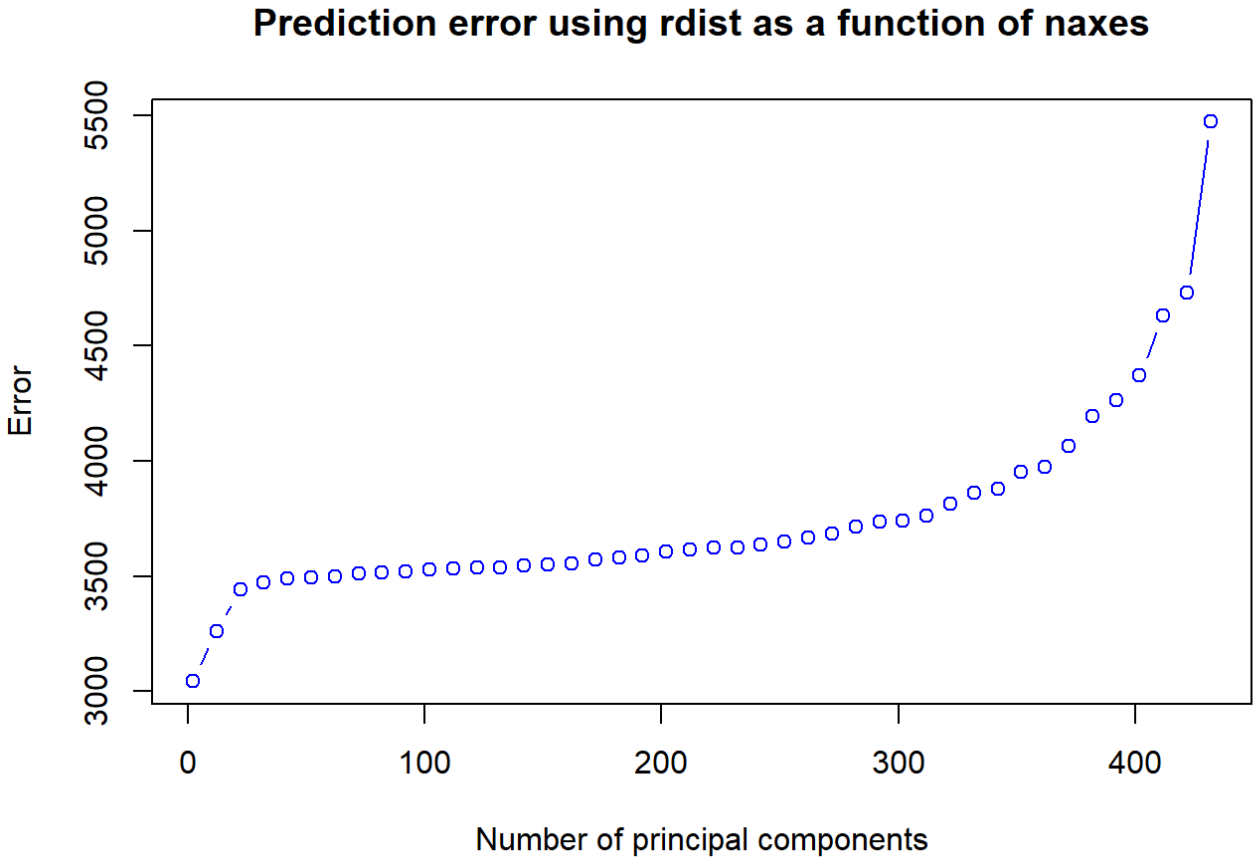
```
## [1] 3156.903
```

\(\textbf{c)}\)

```
#Allows to compute the mean training error for all iteration of
#10-folds cross validation
cv_train_error <- function(naxes){
  errors <- c()
  for (k in 1:10){
    errors <- c(errors, training_error(k, naxes))
  }
  return(mean(errors))
}

#Displays the training and prediction errors as functions of naxes,
#the number of principal components used in the model
display_errors <- function(){
```

```
  values_of_axes <- c()
  train_errors <- c()
  predict_errors <- c()
  for (naxes in seq(2, 440, by = 10)){
    predict_errors <- c(predict_errors, rd_predict_error(naxes))
    train_errors <- c(train_errors, cv_train_error(naxes))
    values_of_axes <- c(values_of_axes, naxes)
  }

  plot(values_of_axes, train_errors, xlab = "Number of principal components",
       ylab = "Error", main = "Training error as a function of naxes",
       type ="b", col ="red")
  plot(values_of_axes, predict_errors, xlab = "Number of principal components",
      ylab = "Error", main = "Prediction error using rdist as a function of naxes"
,
       type ="b", col ="blue")
}
display_errors()
```





\(\textbf{d)}\) To choose the best model, we have to take into consideration 2 constraints: the first one is to choose the one with the smallest prediction error and the second one is to avoid to have an infinite variance which will cause a multicolinearity in our model.

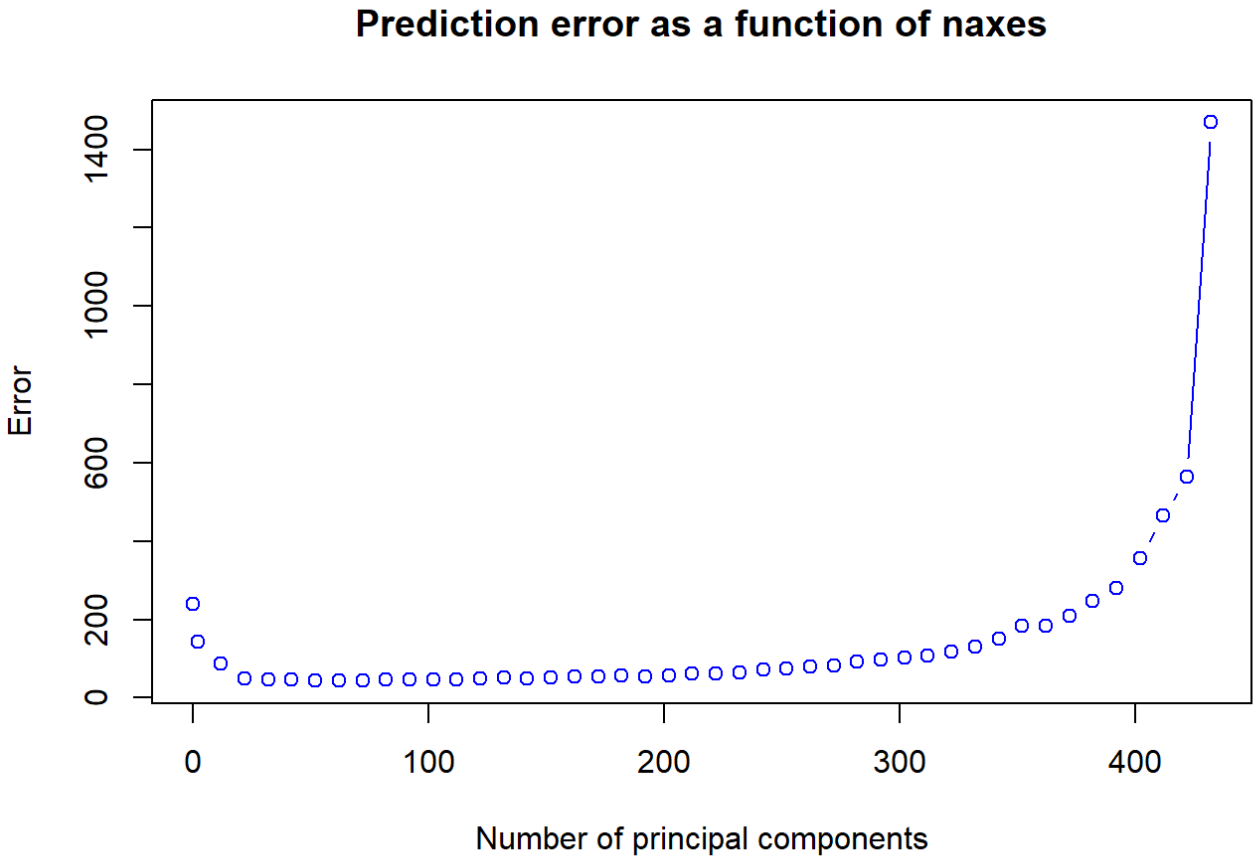
As we feared in the previous part, the overfitting problem is clearly visible in the representation of the prediction error, and the training error only decreases, which explains the good (but erroneous) results encountered above. Predicting the geographic origins of a population from these genetic markers in another database will therefore not be optimal with the 250 principal components model.

We also see that the prediction error seems to stabilize at an acceptable value around 100 main components, while having a minimal difference with the training error which is also acceptable. We then consider the model with 100 principal components. With this choice we will have an accepted prediction error and a nice variance of the model.

Note however that the prediction error, unlike the training error, does not have the expected form. Instead of decreasing to reach a minimum and then increasing again due to the overfitting problem, it only increases according to the complexity of the problem and its minimum is reached for small values of the number of principal components. This is due to the use of the \(\texttt{rdist}\) function which, as explained above, combines several errors. To validate our choice, let's look at the error prediction calculated in a more "traditional" way:

```
#Allows to calculate the prediction error of the model using a
#certain validation set
#The prediction error would be the mean of the errors of both predictions for
#lat and long
prediction_error <- function(number_valid_set, naxes){
  valid <- belongs_vector[1:494] == number_valid_set
  if(naxes == 0){
    lmlatcv <- lm(NAm2[!valid,7]~1, data=NAm2)
    lmlongcv <- lm(NAm2[!valid,8]~1, data=NAm2)
    longpredict <- predict(lmlongcv, newdata=pcaDf[valid,])
    latpredict <- predict(lmlatcv, newdata=pcaDf[valid,])
  }
  else{
    lmlatcv <- lm(NAm2[!valid,7]~., data=pcaDf[!valid,1:naxes])
    lmlongcv <- lm(NAm2[!valid,8]~., data=pcaDf[!valid,1:naxes])
    longpredict <- predict(lmlongcv, newdata=pcaDf[valid,1:naxes])
    latpredict <- predict(lmlatcv, newdata=pcaDf[valid,1:naxes])
  }
  error_long <- mean((NAm2[valid,8]-longpredict)^2)
  error_lat <- mean((NAm2[valid,7]-latpredict)^2)
  predict_error <- mean(error_long, error_lat)
  return(predict_error)
}

#Allows to compute the mean prediction error for all iteration of
#10-folds cross validation
cv_predict_error <- function(naxes){
  errors <- c()
  for (k in 1:10){
    errors <- c(errors, prediction_error(k, naxes))
  }
  return(mean(errors))
}

#Displays the prediction error not using rdist
#as a function of the number of principal components
display_predi_error <- function(){
  #We use the case of 0 predictors as well
```

```
  values_of_axes <- c(0)
  predict_errors <- c(cv_predict_error(0))
  for (naxes in seq(2, 440, by = 10)){
    predict_errors <- c(predict_errors, cv_predict_error(naxes))
    values_of_axes <- c(values_of_axes, naxes)
  }

  plot(values_of_axes, predict_errors, xlab = "Number of principal components",
      ylab = "Error", main = "Prediction error as a function of naxes",
      type ="b", col ="blue")

  {
  cat("The prediction error reaches its minimum around the value of 60 principal c
omponents :", min(predict_errors), "\n")
  cat("The prediction error for the value of 100 principal components is :",
      cv_predict_error(100), "\n")
  cat("The prediction error for the value of 120 principal components is :",
      cv_predict_error(120), "\n")
  cat("The prediction error for the value of 150 principal components is :",
      cv_predict_error(150), "\n")
  }
}

display_predi_error()
```




```
## The prediction error reaches its minimum around the value of 60 principal compo
nents : 43.61836
## The prediction error for the value of 100 principal components is : 45.48846
```

```
## The prediction error for the value of 120 principal components is : 48.4755
## The prediction error for the value of 150 principal components is : 51.78446
```

The curve obtained in this case is more expected. We notice that the prediction error reaches its minimum around a value of 60 principal components and does not vary enormously until reaching 110 principal components where it begins to increase seriously. This confirms our choice to take the model with 100 principal components, since the prediction error is almost minimal in this case and this allows to have a greater cumulative variance while keeping a prediction error very favorable to a good prediction.

```
#Cumulative variance for the 60 first principal components
summary(pcaNAm2)$importance[3,60]
```

```
## [1] 0.32783
```

```
#Cumulative variance for the 100 first principal components
summary(pcaNAm2)$importance[3,100]
```

```
## [1] 0.4511
```

Thus our choice is \(\textbf{the model with 100 principal components}\). Its prediction error is :

```
#training error for our model:
cv_train_error(100)
```

```
## [1] 26.59305
```

```
#the prediction error for our model:
cv_predict_error(100) #without rdist
```

```
## [1] 45.48846
```

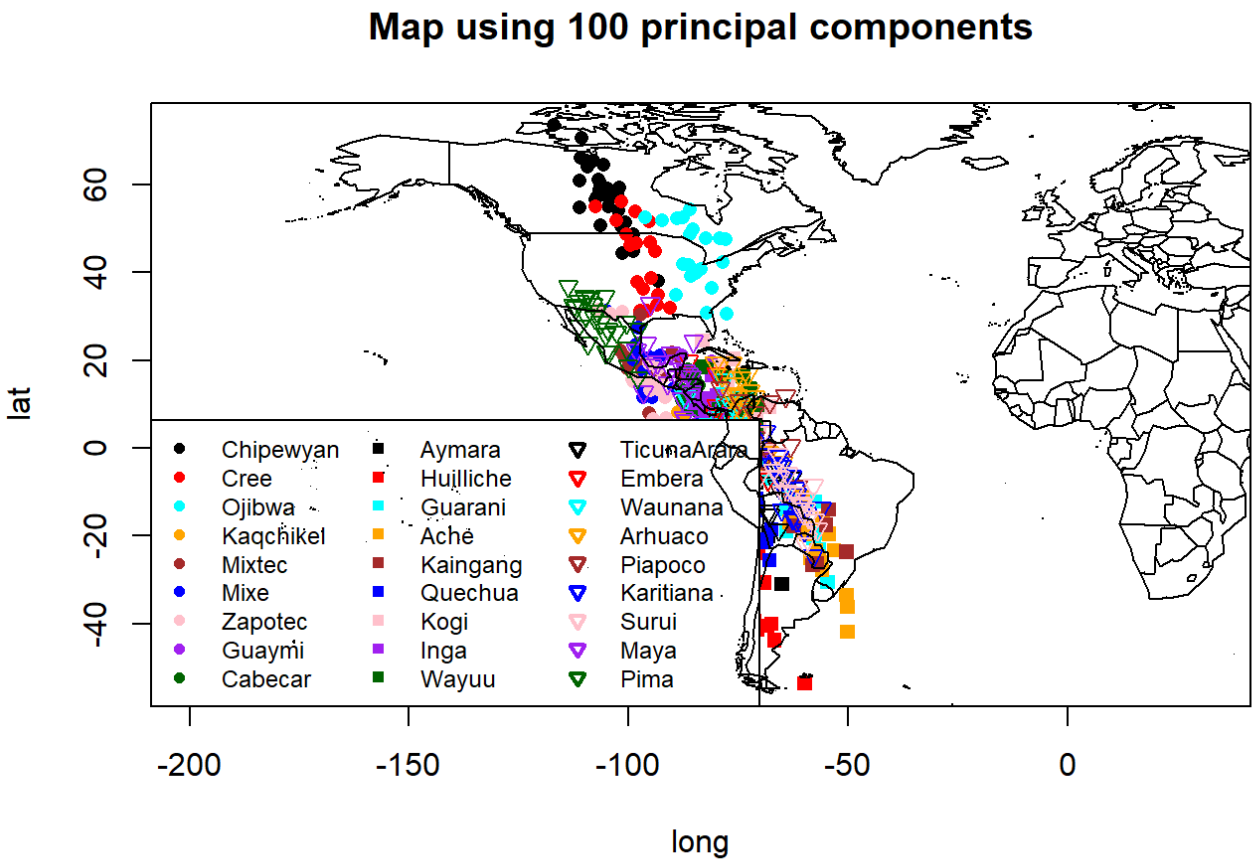
```
rd_predict_error(100) #with rdist
```

```
## [1] 3527.298
```

It’s the closest error to the training error, which means that we did a nice fitting.

```
#Map for our model
pcaDf <- data.frame(pcaNAm2$x)
lmlat <- lm(NAm2[,7]~., data=pcaDf[,1:100])
lmlong <- lm(NAm2[,8]~., data=pcaDf[,1:100])

plot(lmlong$fitted.values, lmlat$fitted.values, col ="white", xlab="long",
     ylab="lat" ,asp =1, main = "Map using 100 principal components")
for ( i in 1: npop ){
  lines(lmlong$fitted.values[which(NAm2[ ,3]==names[i])] ,
        lmlat$fitted.values[which(NAm2[ ,3]==names[i])] ,
        type ="p", col = colPalette[i], pch = pch[i])
```

```
}
legend ("bottomleft",legend =names , col = colPalette , lty = -1 ,
        pch = pch , cex =.75 , ncol =3 , lwd =2)
map ("world",add = T)
```



On this new map, the prediction seems less good than on the previous maps since the points are less grouped together and the training error is greater. But we are now sure thanks to our study that this model will best adapt to the prediction for a population from its genetic markers outside our database.

6. Conclusion

We began our study by attempting a linear regression to predict population geographic location from the initial dataset. But we quickly understood that this was mathematically impossible, as was shown in the second part. We then became interested in the PCA to regroup the genetic markers and thus reduce the number of measurements to be able to carry out a regression. This was the subject of the third part, which thus raised the following question: how many principal components are necessary for a good prediction? At the end of this part, we concluded that to have a cumulative variance of the order of 80%, we would need around 270 principal components. The fourth part is in this order of ideas and uses the first 250 principal components and gives us good results at first sight, except that these results are obtained by a prediction on the training set and that the problem of overfitting can be underlying. This is what we confirmed in the fifth and last part, where we proposed as an improvement the use of the classic prediction error to approve and confirm our choice to take the model with 100 main components, including the prediction error is not far from being minimal, while guaranteeing the smallest difference with the training error and a satisfactory cumulative variance.

```
pcaDf <- data.frame(pcaNAm2$x)
lmlat <- lm(NAm2[,7]~., data=pcaDf[,1:100])
```

```r
lmlong <- lm(NAm2[,8]~., data=pcaDf[,1:100])
regOutLat <- capture.output(summary(lmlat))
regOutLong <- capture.output(summary(lmlong))

#Shows some lines of each summary
{
  print(regOutLat[2], quote = FALSE)
  print(regOutLat[3], quote = FALSE)
  for (i in 9:28){
    print(regOutLat[i], quote = FALSE)
  }
  print(regOutLat[117], quote = FALSE)

  cat("\n")

  print(regOutLong[2], quote = FALSE)
  print(regOutLong[3], quote = FALSE)
  for (i in 9:28){
    print(regOutLong[i], quote = FALSE)
  }
  print(regOutLong[117], quote = FALSE)
}
```

```
## [1] Call:
## [1] lm(formula = NAm2[, 7] ~ ., data = pcaDf[, 1:100])
## [1] Coefficients:
## [1]             Estimate Std. Error t value Pr(>|t|)
## [1] (Intercept)  8.45119    0.41813  20.212  < 2e-16 ***
## [1] PC1          0.15348    0.14058   1.092 0.275605
## [1] PC2          5.71558    0.16334  34.992  < 2e-16 ***
## [1] PC3          3.07596    0.17417  17.661  < 2e-16 ***
## [1] PC4         -1.46982    0.18041  -8.147 5.02e-15 ***
## [1] PC5         -1.06046    0.18897  -5.612 3.78e-08 ***
## [1] PC6         -1.96585    0.20128  -9.767  < 2e-16 ***
## [1] PC7          1.71804    0.21014   8.176 4.09e-15 ***
## [1] PC8         -2.12434    0.21860  -9.718  < 2e-16 ***
## [1] PC9         -0.76612    0.23104  -3.316 0.000998 ***
## [1] PC10        -1.76357    0.23534  -7.494 4.47e-13 ***
## [1] PC11         0.32829    0.24941   1.316 0.188854
## [1] PC12        -0.51983    0.25032  -2.077 0.038486 *
## [1] PC13        -2.17837    0.25674  -8.485 4.46e-16 ***
## [1] PC14        -2.38483    0.26308  -9.065  < 2e-16 ***
## [1] PC15         2.61258    0.27137   9.627  < 2e-16 ***
## [1] PC16         2.58436    0.27399   9.432  < 2e-16 ***
## [1] PC17        -3.59978    0.27942 -12.883  < 2e-16 ***
## [1] F-statistic: 29.39 on 100 and 393 DF,  p-value: < 2.2e-16
##
## [1] Call:
## [1] lm(formula = NAm2[, 8] ~ ., data = pcaDf[, 1:100])
## [1] Coefficients:
## [1]               Estimate Std. Error  t value Pr(>|t|)
## [1] (Intercept) -8.097e+01  2.641e-01 -306.563  < 2e-16 ***
## [1] PC1         -6.200e-02  8.880e-02   -0.698 0.485449
## [1] PC2         -3.853e+00  1.032e-01  -37.338  < 2e-16 ***
```

```
## [1] PC3          -1.827e+00  1.100e-01  -16.608  < 2e-16 ***
## [1] PC4           1.265e+00  1.140e-01   11.097  < 2e-16 ***
## [1] PC5           8.746e-01  1.194e-01    7.327 1.35e-12 ***
## [1] PC6          -1.444e+00  1.271e-01  -11.354  < 2e-16 ***
## [1] PC7          -6.076e-01  1.327e-01   -4.578 6.32e-06 ***
## [1] PC8           1.116e+00  1.381e-01    8.083 7.89e-15 ***
## [1] PC9           1.300e+00  1.459e-01    8.909  < 2e-16 ***
## [1] PC10          6.095e-01  1.487e-01    4.100 5.03e-05 ***
## [1] PC11          9.386e-01  1.576e-01    5.957 5.70e-09 ***
## [1] PC12          5.312e-01  1.581e-01    3.359 0.000859 ***
## [1] PC13         -5.034e-01  1.622e-01   -3.104 0.002049 **
## [1] PC14          2.771e+00  1.662e-01   16.677  < 2e-16 ***
## [1] PC15         -1.928e+00  1.714e-01  -11.244  < 2e-16 ***
## [1] PC16         -9.347e-01  1.731e-01   -5.401 1.15e-07 ***
## [1] PC17          1.052e+00  1.765e-01    5.960 5.61e-09 ***
## [1] F-statistic: 30.18 on 100 and 393 DF,  p-value: < 2.2e-16
```

We thus observe that the predictions made are conclusive and that the choice of the model is quite relevant since in both cases the p-value of the F-test is small. Also, many predictors are significant in predicting longitude and latitude. The model is thus adequate for the desired predictions.