PCA, KNN and FDA implementation

Silvana Alvarez - Sergio Quintanilla

2025-03-12

```
#install.packages("OpenImageR")
library(OpenImageR)
library(dplyr)
```

```
##
## Adjuntando el paquete: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(patchwork)
```

```
source("function_loading.R")
```

Part A

Implement a facial recognizer based on Principal Component Analysis.

Solution:

First we show how we create the DB, then in each section we show the functions that we made to create the PCA, KNN and tune the parameters. Finally, we display the application of all the steps using the tunned parameters.

```
# Files
Files = list.files(path="Training/")

# Labels
labels <- as.numeric(gsub("[^0-9]", "", Files))

# Data Base
X = matrix(NA, nrow=150, ncol = 108000)
```

```r
for(i in seq_along(Files)){
  Im = readImage(paste0("Training/",Files[i]))
  ri=as.vector(Im[,,1])
  gi=as.vector(Im[,,2])
  bi=as.vector(Im[,,3])
  X[i,] = cbind(ri,gi,bi)
}
dim(X)
```

```
## [1]    150 108000
```

With the previous process we obtain a matrix of 150 rows (images), and 108.000 columns, that represents each pixel of the image, in each one of the RGB colors.

A)

Build a function that implements the Principal Component Analysis. This function takes as input a set of observations and returns the mean of these observations, the matrix P containing the eigenvectors and a vector D containing the variance explained by each principal axis. It is only allowed to use the function eigen.

Solution:

We create a function where we implement the following steps:

```r
PCA
```

```
## function (X_train, n_comp)
## {
##     M <- colMeans(X_train)
##     G <- X_train - M
##     G_ <- t(G)
##     n_train <- nrow(X_train)
##     Sigma_s <- (G %*% G_)/(n_train - 1)
##     eig = eigen(Sigma_s)
##     e_vec_s = eig$vectors
##     P = G_ %*% e_vec_s
##     L = diag(eig$values)
##     D = round(eig$values/sum(eig$values), 3)
##     top_comp <- P[, 1:n_comp]
##     list(components = top_comp, mean = M, vectors = P, values = eig$values,
##         var_exp = D)
## }
```

Then a small one to project the train and test data, taking into account the PCA made over the train data before:

```r
project_pca
```

```
## function (data, pca_model)
## {
##     centered_data <- data - pca_model$mean
##     projected_data <- as.matrix(centered_data) %*% pca_model$components
##     return(projected_data)
## }
```

B)

Built a classifier (function) that takes as input an image and an object with the parameters of the classifier. Internally, the function uses a k-nn classifier and the PCA representation of the images. If the person in the image belongs to the database, it returns the person's identifiers. Otherwise it returns 0. In order to build this, you will need to consider:

- The percentage of the variance retaining by the PCs
- The number of neighbors of the k-nn
- The similarity metric
- The threshold to determine when the person belongs to the database

Solution:

To begin, we define a special function to define the train/test split.

First we take some labels out in order to define the "impostors" and be allowed to test the threshold to determine when the person belongs to the database.

With the remaining labels we take some images to the test split (that are also in the train) in order to train the KNN as is normally done.

Finally we join the observations obtained in the two previous steps and define them as the test dataset. The remaining observations are going to be the train dataset.

```
create_train_test_split
```

```
## function (data, labels, num_persons_out, split_seed = NULL)
## {
##     labels <- as.character(labels)
##     unique_persons <- unique(labels)
##     if (!is.null(split_seed)) {
##         set.seed(split_seed)
##     }
##     test_persons <- sample(unique_persons, num_persons_out)
##     remaining_persons <- setdiff(unique_persons, test_persons)
##     test_indices_p1 <- which(labels %in% test_persons)
##     remaining_indices <- which(labels %in% remaining_persons)
##     if (!is.null(split_seed)) {
##         set.seed(split_seed)
##     }
##     test_indices_p2 <- sample(remaining_indices, round(length(remaining_indices) *
##         0.2))
##     test_indices <- c(test_indices_p1, test_indices_p2)
##     test_data <- data[test_indices, ]
##     test_labels <- labels[test_indices]
##     train_data <- data[-test_indices, ]
##     train_labels <- labels[-test_indices]
##     return(list(train_data = train_data, train_labels = train_labels,
##         test_data = test_data, test_labels = test_labels, test_indices = test_indices))
## }
```

With that done, we define the KNN for different possible types of distances (Mahalanobis, Euclidean, SSE Modified and W angle) (creo que hay que poner cita y no se si imprimir las distancias o ponerlas en latex).

```
knn_classifier
```

```
## function (train_data, train_labels, test_data, k, percent_threshold,
##     pca_model = NULL, n_comp = NULL, distance_func = mahalanobis_distance)
## {
##     within_distances <- c()
##     between_distances <- c()
##     distances <- matrix(NA, nrow(train_data), nrow(train_data))
##     for (i in 1:nrow(train_data)) {
##         test_point <- train_data[i, ]
##         distances[i, ] <- distance_func(test_point, train_data,
##             pca_model, n_comp)
##     }
##     n <- nrow(train_data)
##     for (i in 1:(n - 1)) {
##         for (j in (i + 1):n) {
##             if (train_labels[i] == train_labels[j]) {
##                 within_distances <- c(within_distances, distances[i,
##                   j])
##             }
##             else {
##                 between_distances <- c(between_distances, distances[i,
##                   j])
##             }
##         }
##     }
##     max_within <- max(within_distances)
##     q10_between <- quantile(between_distances, 0.1)
##     if (max_within < q10_between) {
##         threshold <- max_within + percent_threshold * (q10_between -
##             max_within)
##     }
##     else {
##         threshold <- quantile(within_distances, 1 - percent_threshold)
##     }
##     cat("Threshold value:", threshold, "\n")
##     predictions <- character(nrow(test_data))
##     min_distances <- numeric(nrow(test_data))
##     for (i in 1:nrow(test_data)) {
##         test_point <- test_data[i, ]
##         distances_test <- distance_func(test_point, train_data,
##             pca_model, n_comp)
##         min_dist <- min(distances_test)
##         min_distances[i] <- min_dist
##         if (min_dist > threshold) {
##             predictions[i] <- "0"
##             cat("Sample", i, "distance", min_dist, "> threshold",
##                 threshold, "→ label 0\n")
##         }
##         else {
##             neighbors <- order(distances_test)[1:k]
##             neighbor_labels <- train_labels[neighbors]
##             label_counts <- table(neighbor_labels)
##             most_common_idx <- which.max(label_counts)
##             predicted_label <- names(label_counts)[most_common_idx]
##             predictions[i] <- as.character(predicted_label)
##             cat("Sample", i, "distance", min_dist, "<= threshold",
##                 threshold, "→ KNN label", predicted_label,
##                 "\n")
##         }
##     }
##     return(list(within = within_distances, between = between_distances,
##         thres = threshold, pred = predictions, min_distances = min_distances))
## }
```

C)

Explain how you have determined the previous parameters

Solution:

D)

Repeat b) but using the initial image representation instead of the principal component representation. Based on your results, decide if you prefer to use the principal component representation or the original representation and justify your decisions.

Solution:

Aplication

```r
# Train test split
split_data <- create_train_test_split(data=X, labels = labels, num_persons_out = 2,
                        split_seed = 782)

train_data <- split_data$train_data
train_labels <- split_data$train_labels

test_data <- split_data$test_data
test_labels <- split_data$test_labels

# PCA model with n_comp found
pca_model <- PCA(train_data, results$n_components)

# Projection of train and test
train_pca <- project_pca(train_data, pca_model)
test_pca <- project_pca(test_data, pca_model)

predictions <- knn_classifier(train_data = train_pca, train_labels = train_labels,
                              test_data = test_pca,   k = results$k,
                              percent_threshold = results$percent_threshold,
                              pca_model = pca_model,  n_comp = n_comp)
```

PART B

Implement a facial recognizer based on Fisher discriminant analysis.

A)

Build a function that implements the Fisher Discriminant Analysis. This function takes as input a set of observations and returns the mean of these observations, the matrix P containing the eigen vector of the appropriate matrix and a vector D containing the variance explained by each fisher discriminant. It is only allowed to use the function eigen.

Solution:

B)

Built a classifier (function) that takes as input an image and an object with the parameters of the classifier. Internally, the function uses a k-nn classifier and the Fisher discriminant analysis representation of the images. If the person in the image belongs to the database, it returns the person's identifiers. Otherwise it returns 0. In order to build this, you will need to consider:

- The percentage of the variance retaining by the Fisher discriminant dimensions
- The number of neighbors of the k-nn
- The similarity metric
- The threshold to determine when the person belongs to the database

Solution:

C)

Explain how you have determined the previous parameters

Solution: