EIGENFACES 2.0

DEGREE IN DATA SCIENCE AND ENGINEERING

Alejandro Pérez

Christopher Manzano

04.12.2020

STATISTICAL LEARNING

INTRODUCTION

The goal of this assignment is to apply the new concepts of Multiclass Fisher Linear Discriminant Analysis seen in class to recognize faces from a database. To approach this problem we will first create the different functions we will need, such as PCA or LDA. Later we will build and optimize our KNN classifier needed in the second part of the assignment.

Finally, we will create the main classifier that internally uses the previously created functions to properly classify the pictures. Once this is done we will use cross-validation to check optimal values to get the best possible accuracy in our model. We will take into account things like the number of K neighbors, the number of eigenvectors, the threshold, or the different distance methods.

Note: We will visualize our different results with plots from ggplot2 and scatterplot3d.

PROCEDURE

Before anything, we loaded all the libraries we were going to use: ggplot2, OpenImageR, doParallel, scatterplot3d, tictoc.

Additionally, in order to work with these images, we must first make our images be vectors. For this, we created the vectorize.images function.

Exercise A

The first thing we noticed is that we could not use LDA directly on the converted images, as their dimension when computing the LDA would be impossible to handle by our computers. To work this around we decided to run PCA on the images so as to greatly reduce their dimensions, and so we created the function pca(X, center = T, scale = T) which returns the Principal Components (PCs), as well as the eigenvectors and eigenvalues.

Then, we created the function mda(data, labels) (m for multiclass) to compute the LDA that internally calculates the S-within (SW) and S-between (SB) matrices to then return the corresponding eigenvectors and eigenvalues.

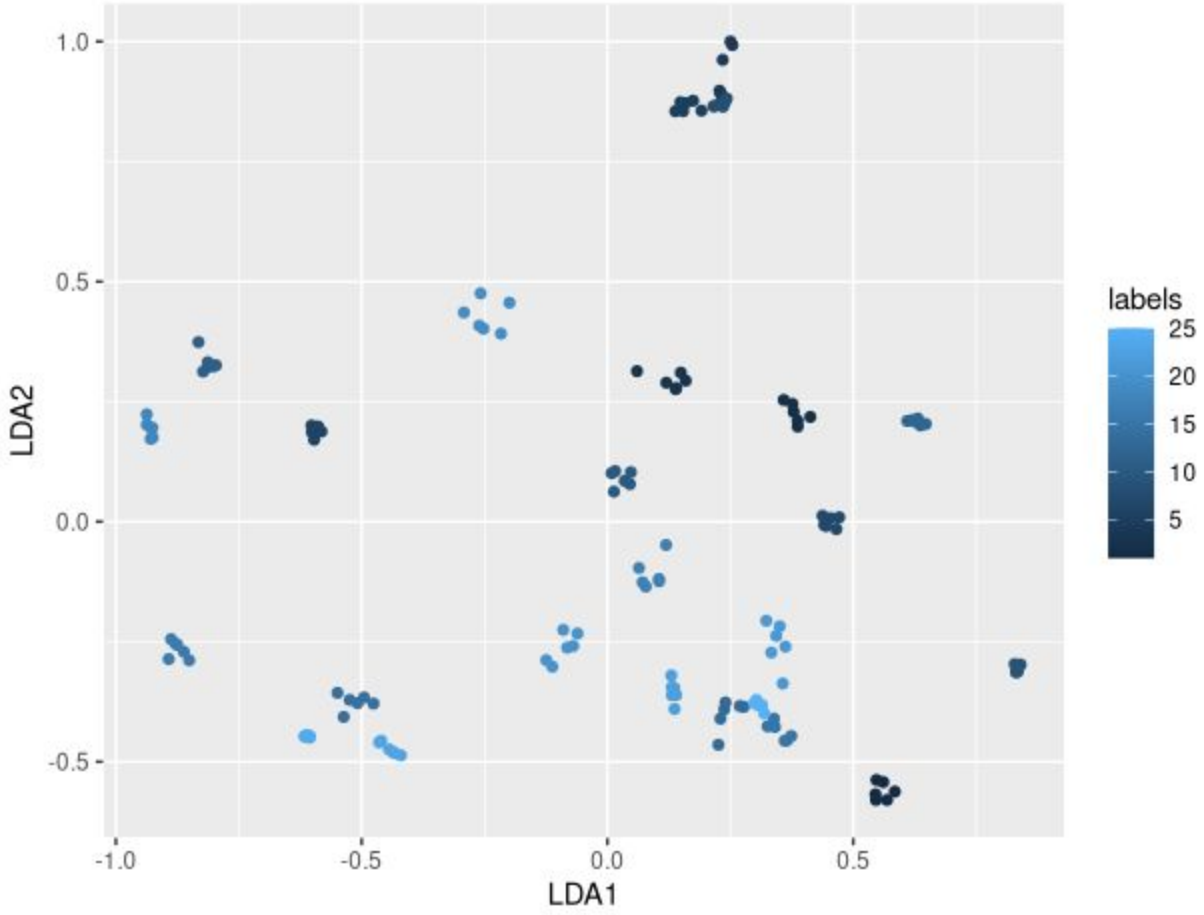
From this first step, we got the following table:

Note: We decided to keep 12 principal components from pca (around 75 percent of variance) because we were struggling with complex numbers and we did try to avoid overfitting.

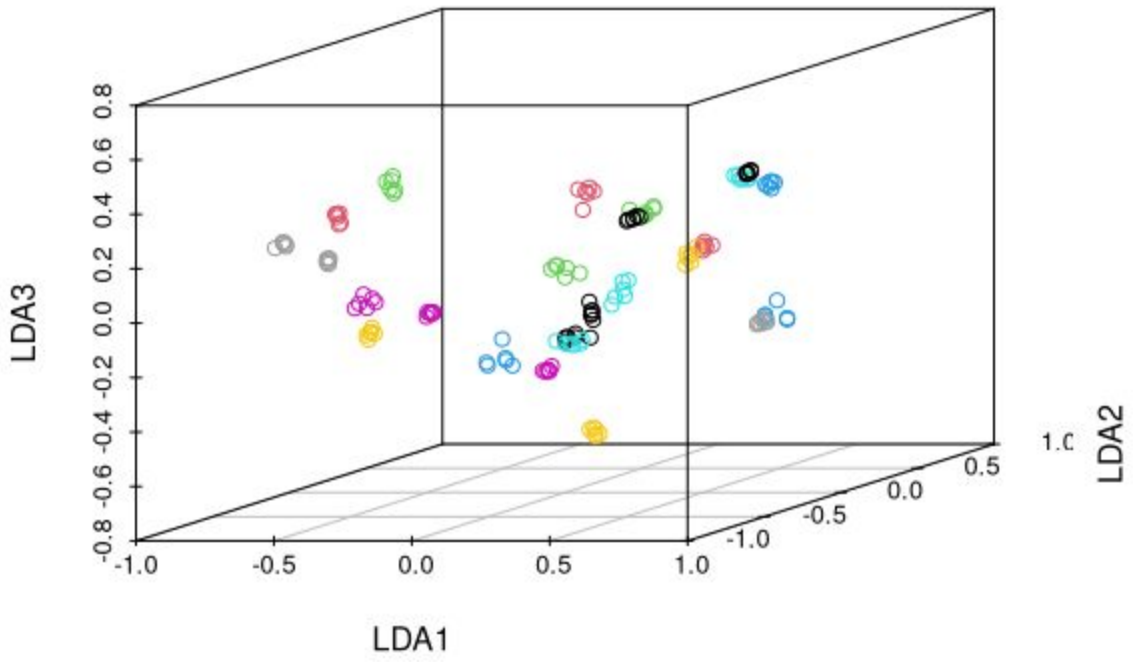
Eigenvectors vs Variance LDA

N	1	2	3	4	5	6	7	8	9	10	11
Var	45	66	77	84	90	93	96	98	99	100	100

To visualize how the above functions are capable of separating the data and classify the different people, we created some plots computing the LDA from (in this case) the first 12 eigenvectors we obtained from the prior PCA.



In this plot, we used the first two eigenvectors from the LDA and we can see that the dots are mostly grouped in the same color, but in some cases some overlapping occurs. Because of this, we chose to visualize the data with the first three eigenvectors from the LDA and used scatterplot3d to plot it since ggplot can't plot three dimensions.



Note: the above plot has scaled data so as to visualize the results with more user-friendly numbers.

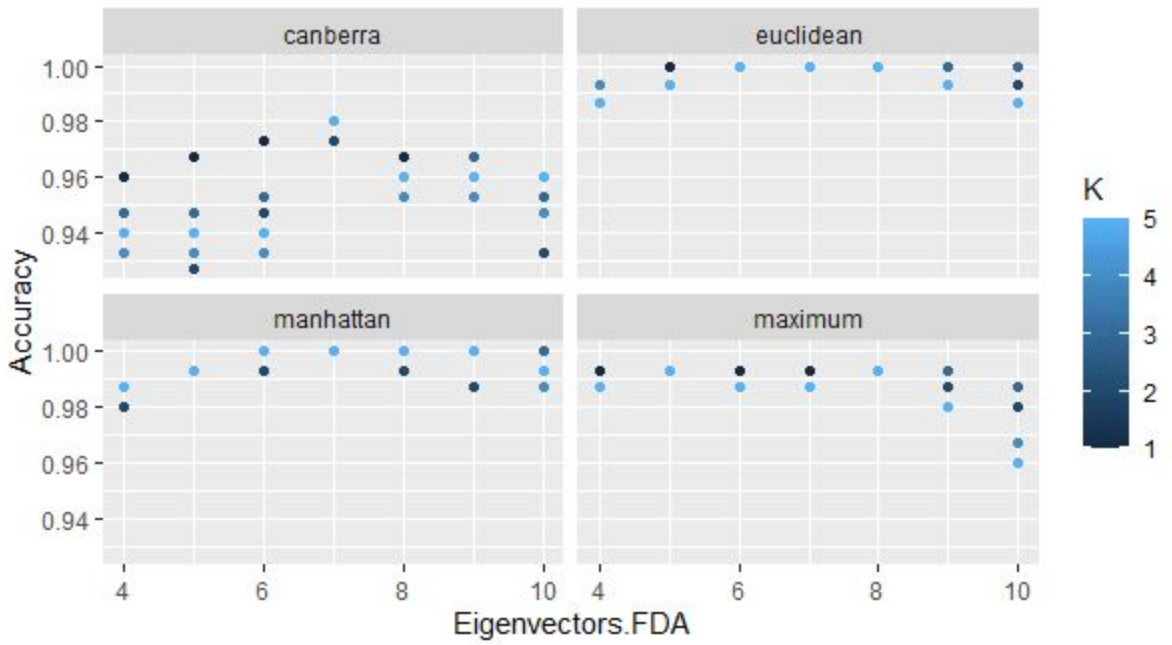
Exercise B

In this second part of the assignment, we built the knn.classifier(parameters, newdata) using part of the code we were given with some tweaks to optimize its performance. The parameters must specify the train data, the distance method we wish to use, and also the preferred value for the k. This knn classifier returns a matrix containing the results with every computed distance.

Next, we created our main classifier classifier(parameters, new.images), which internally starts by vectorizing the images. Then it uses PCA to reduce the dimensionality projecting the data to the first 12 eigenvectors (containing around 95 % of variance). Then, we perform LDA on the train set with their labels and project both, the train set and the new set to the first n eigenvectors (nmda) obtaining the final matrix. Finally, the classifier computes and returns the Knn applied to the previous data with the given parameters.

For the cross-validation we took into account the different distances (manhattan, euclidean, canberra, maximum), values for the K (ranging from 1 to 5), the threshold, and the n first eigenvectors we obtain from the LDA (ranging from 4 to 10). From here we obtain some predictions which tell us the Accuracy obtained for every combination of the various parameters (distance method, k, threshold, etc.). We save these predictions in a ".RData" file and proceed to check the plots to decide on what the best combination of parameters is.

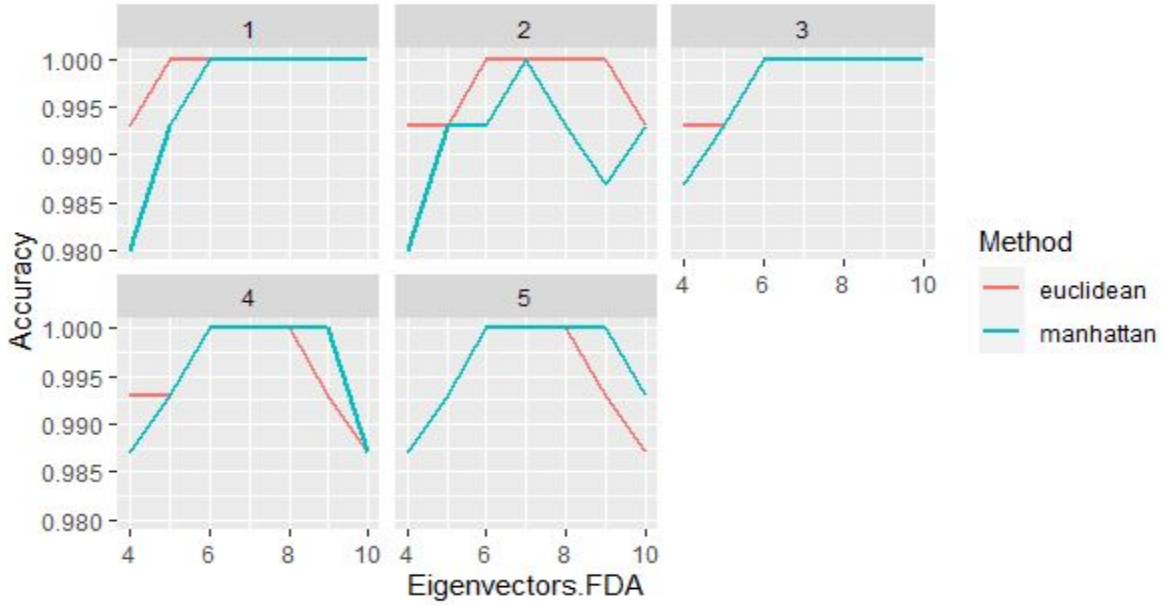
In our first approach, we did not consider the threshold and just focused on the other parameters.



In this plot, we can easily see that the only two distances that really achieve 100% accuracy are manhattan and euclidean. Our main objectives are minimizing the complexity and maximizing

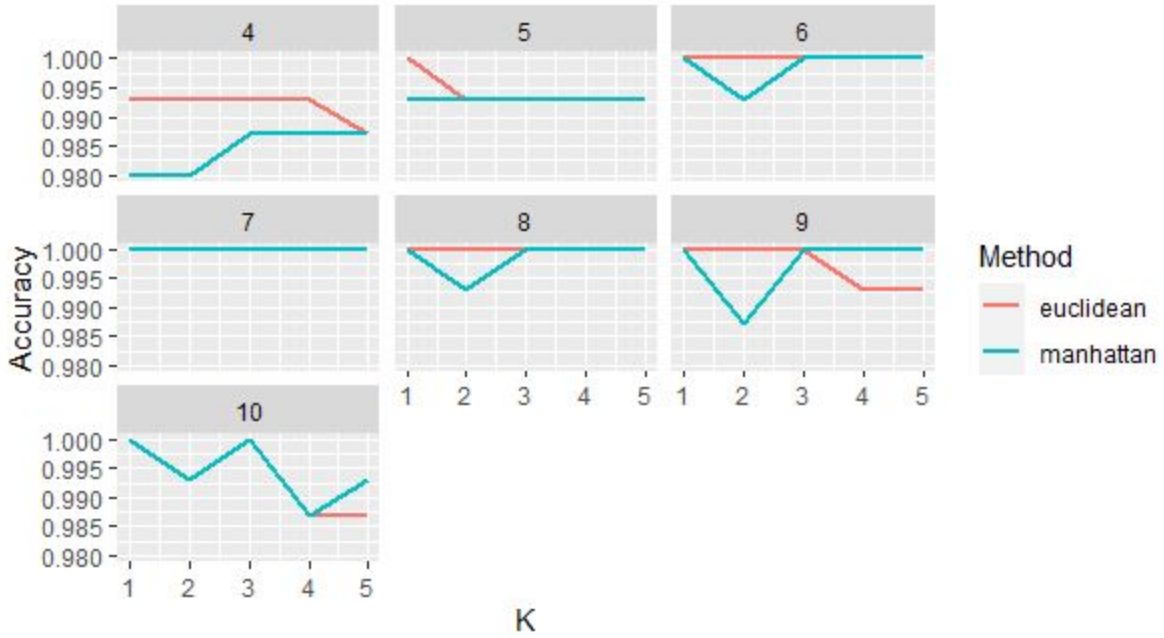
the accuracy of our model. We are going to find a simple combination of eigenvectors and k.

The following plot contains the accuracy as a function of the eigenvectors, for every value of k:



Here we see that k = 1 gives 100% accuracy if we use more than five eigenvectors with the euclidean distance. K = 1 and k = 3 give the most stable lines. And that the euclidean distance is more stable in terms of accuracy than the manhattan distance.

The following plot contains the accuracy as a function of k, for every eigenvector:

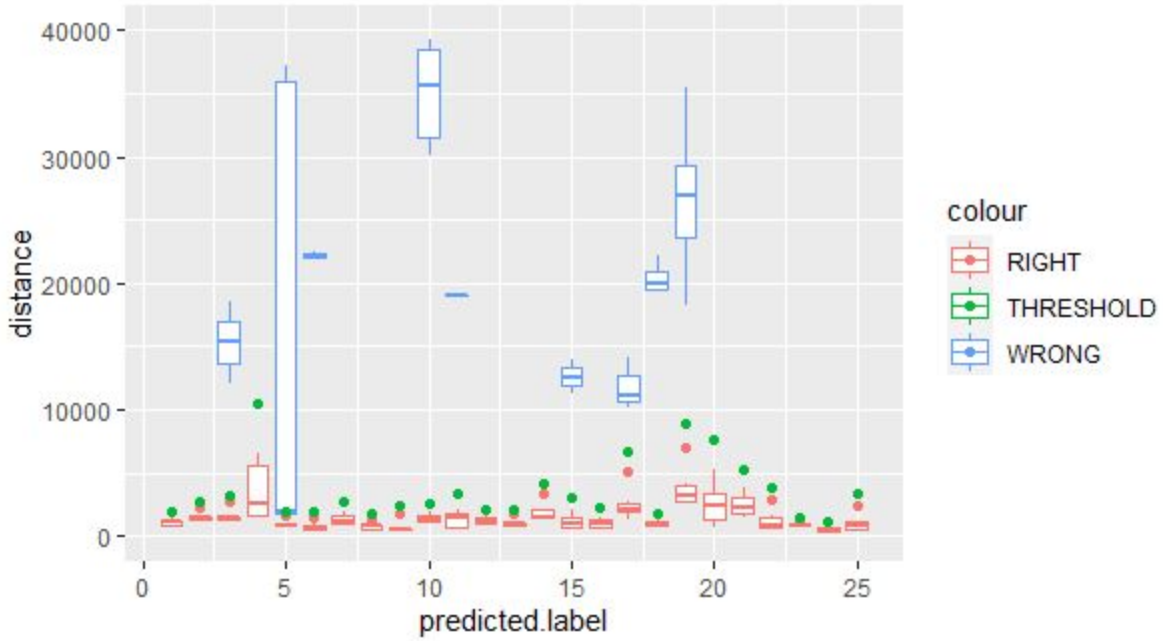


Here we can observe that at the 7 eigenvectors (approx. 96 % of variance) it does not matter which k or method we use. However, with 6 (approx. 93 % of variance) the only constraint is not choosing K = 2 and manhattan distance. Meanwhile picking 5 looked a bit risky.

Our winners were 6 eigenvectors, K = 1, and the euclidean method for the distance.

We ran another loop adding some impostors (images labeled as 0) and saved the distances in order to determine the threshold.

Our threshold per class is the mean distance of the correctly classified images plus three standard deviations (trying to emulate the empirical rule for the normal distribution).



CONCLUSION

Through the making of this model, it became clear to us that there were quite a few things that needed to be understood before facing this problem. Nonetheless, we are beginning to understand that even though it may seem “boring” to learn all the underlying mathematical principles in the model we built, the results have unimaginable potential, and thus we must “keep our eyes on the prize”.

In this world we all take part in, the use of facial recognition programs and algorithms is becoming more and more demanding, and since people’s face is just as unique as their fingerprint, it could well become our very own unique password.