Sofía Pérez Pérez and Paula Serna del Amo

PROJECT 2- Fisher Discriminant Analysis

The goal of this second part of the project is to build a new classifier such that given a photo, it is able to determine whether it belongs to one person of our dataset, and if so, which person. However, we will not use the same transformations as we did in the previous part, or at least not only those ones. In the first project, we used the PCA technique in order to reduce dimensions while retaining enough variance. Nonetheless, this time we will apply PCA, and in addition we will compute a fisher discriminant analysis.

To begin with, we started reading our dataset of images as we did in the last part, using the function list.files(), and then creating a matrix in where each row is a photo of a person. Since our data frame contains an enormous number of variables, the total number of pixels (108000), it is extremely useful to compute PCA in order to reduce dimensionality, and also because we would not be able to compute the fisher discriminant analysis. As it turned out in the first project, we are not able no compute PCA in the standard way, since the number of observations is smaller than the number of pixels. Therefore, we used a trick explained in the first part that enables us to obtain a new dataset with at maximum 150 variables, that is, the number of observations. After this step is completed, we included a new transformation, called the fisher discriminant analysis, that consist of projecting the points into one line, so they are separated by each class, that is, by each person. Apart from that, in order to compute the "best" projection, we will try to maximize the distance between different groups, thus the mean of each class, while minimizing the variance of each group, hence the dispersion. To achieve this step, we first needed to compute the mean of each class, and adding them all with the formula,

$$S_B \sum_{i=1}^{k} n_i (m_i - m)(m_i - m)^T$$

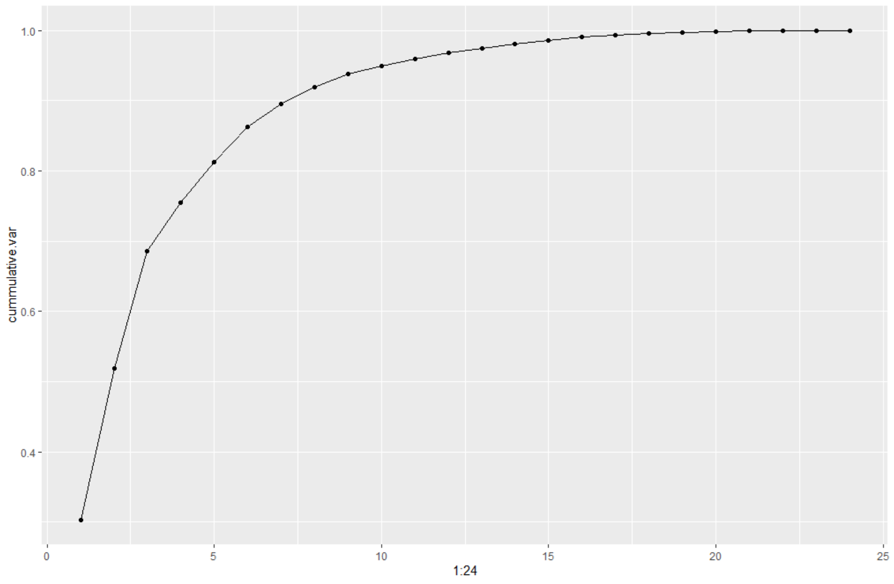
being m the total mean of the data, we computed the matrix SB.

Then, we needed to make use of the matrix SW and of the scatter matrices that were computed with the formula,

$$S_W=\sum_{i=1}^{k} S_i=\sum_{i=1}^{k}\sum_{j\in c_i}(X_j-m_i)(X_j-m_i)^T$$

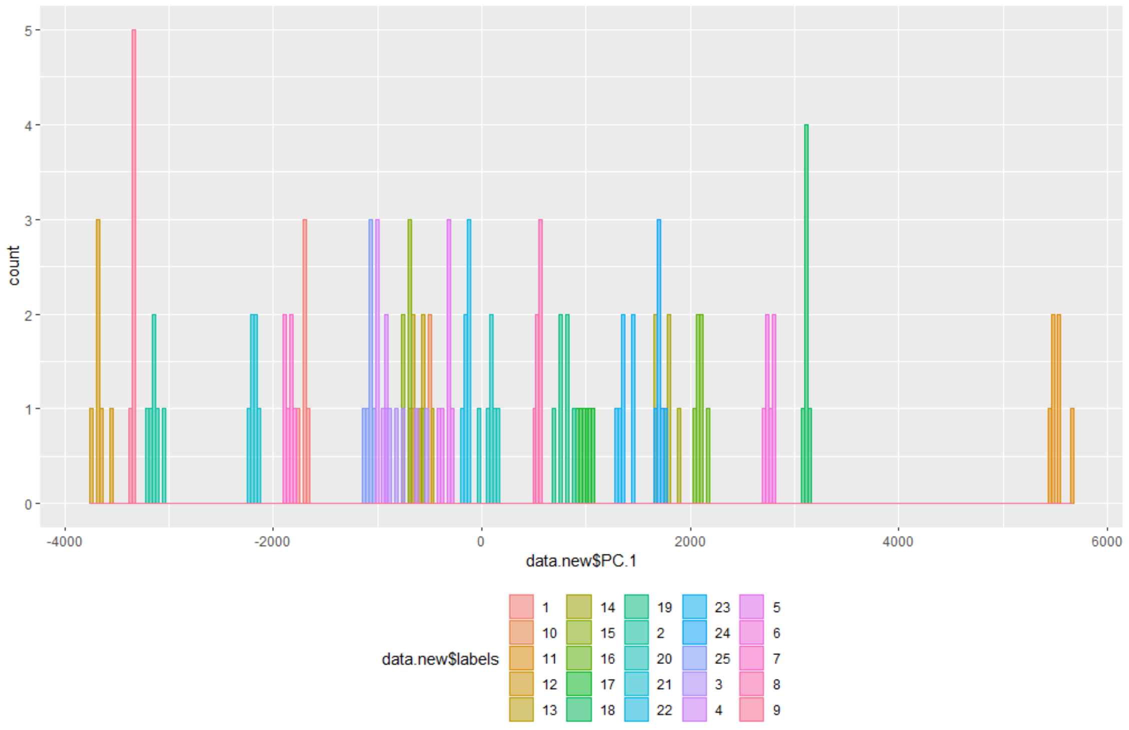
Finally, the projection line we are looking for coincides with the eigenvectors of the matrix Sw^{-1}SB. We want to build a scalar function that is large when the between class covariance matrix is large and small when the within class covariance matrix is small. If we had had only two labels, there would only be one nonzero eigenvector. However, in our case we are dealing with several eigenvectors (in concrete, 10), so we are going to choose the ones that provide us with a reasonable variance of the original data.

One aspect that we needed to take into account is that we had to take at maximum 25 eigenvectors (number of classes) in the PCA part since with more, the fisher part would not work correctly, as it would obtain imaginary numbers in the process of calculating the FDA eigenvectors.



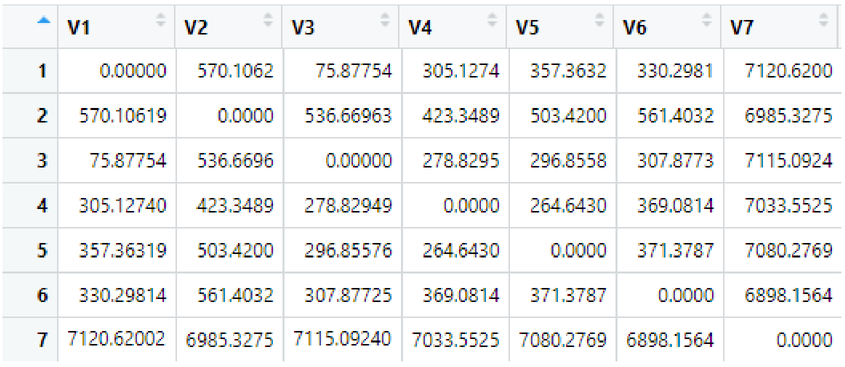
With our training data, we obtained that with the 10 largest eigenvectors of $Sw^{-1}SB$ (symmetric and positive-semidefinite matrix) we would be retaining 95% of the variance. However, we will use cross validation to compute a more accurate value.

Once we have computed the line and chosen the number of variables of the new data, in order to get a more visual idea of what we are doing, we project the observations into the first two.



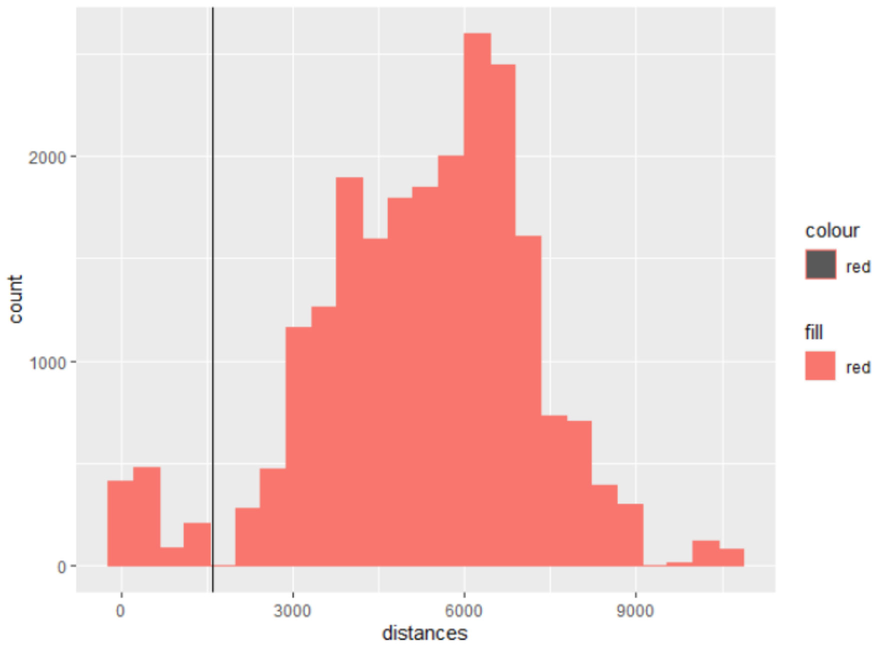
As it can be seen, every observation belonging to one person is grouped with the rest of the same one, ending up with 25 groups of points. In addition, each group follows a Gaussian distribution, as it can be perfectly seen in the second one starting from the right.

Once PCA is performed, we calculated a matrix of distances, which would be used later to determine the threshold, hence, the distance which is going to analyse whether a person belongs to the dataset or not. As we explained in the first project, it can be seen that the distances of one person to himself are really small, as seen in the first 6 columns and rows, compared to the rest; which makes sense because for instance, the seventh row corresponds to a different person.



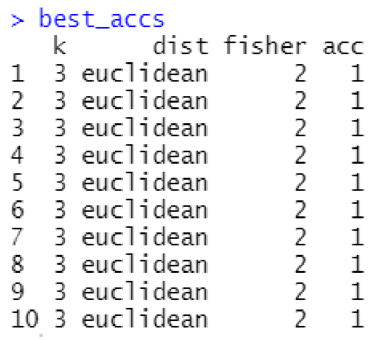
	V1	V2	V3	V4	V5	V6	V7
1	0.00000	570.1062	75.87754	305.1274	357.3632	330.2981	7120.6200
2	570.10619	0.0000	536.66963	423.3489	503.4200	561.4032	6985.3275
3	75.87754	536.6696	0.00000	278.8295	296.8558	307.8773	7115.0924
4	305.12740	423.3489	278.82949	0.0000	264.6430	369.0814	7033.5525
5	357.36319	503.4200	296.85576	264.6430	0.0000	371.3787	7080.2769
6	330.29814	561.4032	307.87725	369.0814	371.3787	0.0000	6898.1564
7	7120.62002	6985.3275	7115.09240	7033.5525	7080.2769	6898.1564	0.0000

Afterwards, we plotted a histogram in which we can observed that from the value 1700 and so on, the distances represented are much bigger than the ones in the left part, which belong to the same person, so, we determined 1700 as the threshold (represented by the black line).



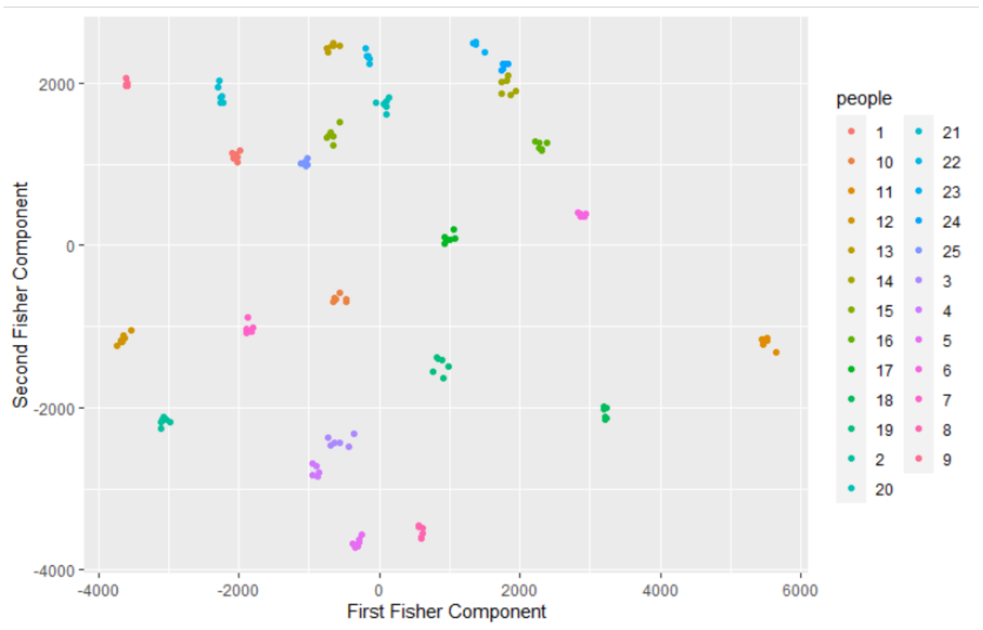
Now, the next step was to create the function which worked as our classifier. Since the standard knn classifier only uses the Euclidean distance and we wanted to test with different distance methods, we had to create it ourselves. We defined it in such a way that we introduce the data projected into the new variables instead of the original data. This function, called KNN, calculates the distances to the person being classified and returns a prediction with the label of the person with minimum distance to it.

In addition, we needed to train the algorithm in order to obtain the optimal parameters for our function. These are k (the number of neighbours), the distance method, and the number of eigenvectors we would use from the fisher data. However, we already found those values then when calculating the eigenvectors of $Sw^{-1}Sb$. We performed k-fold cross-validation as we considered it was the best method to achieve a good result, and inside it we included an already built data frame called "best_accs" whose function was to keep the parameters that provide the maximum accuracy in each fold. This way, we obtained that the maximum accuracy in each fold is obtained with k=3, the Euclidean distance and with 2 eigenvectors.



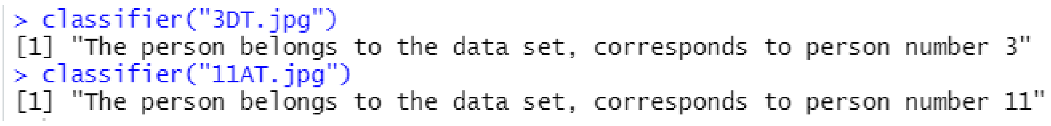
```
> best_accs
   k      dist fisher acc
1  3 euclidean      2   1
2  3 euclidean      2   1
3  3 euclidean      2   1
4  3 euclidean      2   1
5  3 euclidean      2   1
6  3 euclidean      2   1
7  3 euclidean      2   1
8  3 euclidean      2   1
9  3 euclidean      2   1
10 3 euclidean      2   1
```

Although we considered that obtaining accuracies of 1 is not completely correct, and that 2 eigenvectors were too few, we created a plot in which we represented the first 2 principal components (Fisher components) of the projected data (since it was the number of components we obtained). We can observe that people are correctly grouped (by colours) and they are not dispersed, which means that with two components we achieve the correct separation and it is not necessary to take more.



Once we had the optimal parameters that provided us with the best accuracy, we needed to save them as well as other data such as the one containing the number of eigenvectors obtained before (‘best_data’), the matrix P which will project our image into the principal axes, and the mean of the dataset, among others, that will be necessary to carry out our classification task. We did this by creating a .RData file and loading it in our next function, called “classifier”. This function receives an image and standardizes it, subtracting the mean; creates a new vector, projected into the new axis (by means of the P matrix mentioned before), and then this vector is projected into the Fisher axis, using the appropriate number of eigenvectors obtained (in our case, 2).

After, when we had the projected image using the knn function created before, we obtained a prediction of this new vector, but we had to check if the distance between the person predicted and the real person was small enough to be considered the same person. How do we check this? By using the threshold value we already computed; being a condition that if the distance between them is larger than the threshold, they are not the same person, and if it is smaller, they are.



```
> classifier("3DT.jpg")
[1] "The person belongs to the data set, corresponds to person number 3"
> classifier("11AT.jpg")
[1] "The person belongs to the data set, corresponds to person number 11"
```

To conclude, after doing these two projects about PCA and FDA, we wanted to share our point of view about the two of them. Both have benefits and drawbacks; it is difficult to say which one is better. On the one hand, as with FDA you are using much more information (labels included) than when using PCA, and it is a supervised algorithm, we are supposed to obtain a better separation of the classes. But on the other hand, FDA would not be possible without having previously done PCA, since lots of tricks and tools that we do not know would be needed. In addition, PCA reduces the dimensionality of the problem to something that is much more comfortable and once that these dimensions are small enough, you can apply FDA. We could say that FDA depends on another algorithm while PCA does not. Therefore, if we had to choose, we would say PCA is a better option to work with rather than FDA.