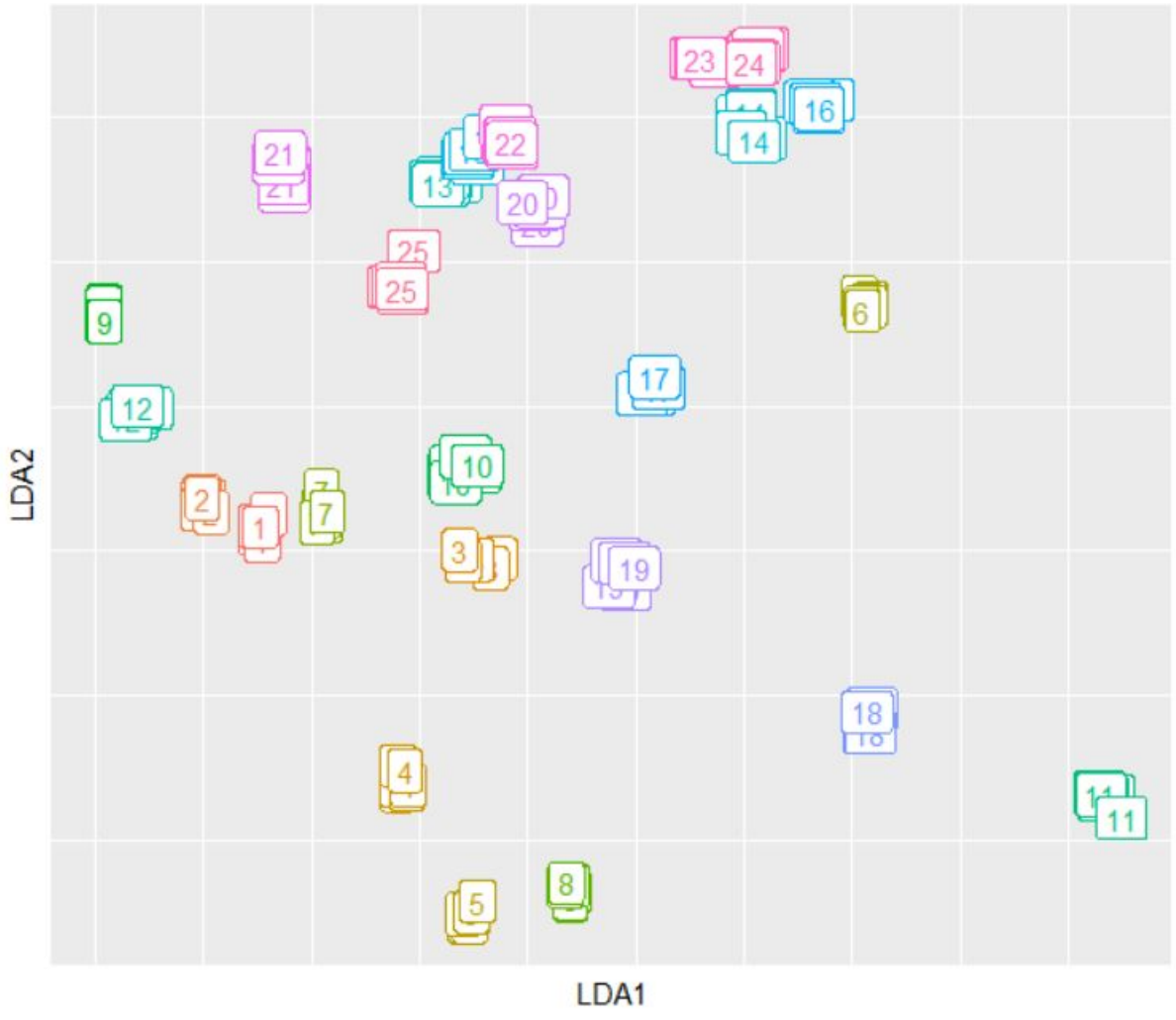
uc3m | Universidad Carlos III de Madrid

Assignment 2: FisherFaces

Statistical Learning

Group 96-97

Aya Ben Hriz and Alejandro Bermejo Gordillo



Introduction

Face recognition may be one of the most mainstream uses of computer vision. Its rapid development has to be attributed to law enforcement agencies all over the world to help identify criminal suspects. From a more general user perspective, it is used to group photos by who is on them or as a security measure to unlock our devices [1].
More formally, we can formulate face recognition as a classification task, where the inputs are images and the outputs are people's names. We're going to discuss a popular technique for face recognition called **Fisherfaces** [3]. And at the heart of Fisherfaces is an unsupervised dimensionality reduction technique called **principal component analysis (PCA)**, and **Fisher Discriminant Analysis for several populations** and we will see how we can combine and apply these general techniques to our specific task of face recognition.

A - Fisher Discriminant Analysis function

The discriminant analysis is a supervised technique and requires a training dataset with predefined labeled groups.
As an argument the function is getting data which has a last column *labels* containing the information about the classes. This data is gonna be projected onto a lower-dimensional space with good class-separability.
We followed these steps in our function so that we can implement FDA

1) We compute the mean vectors for the different classes from the dataset
2) We compute the scatter matrices (in-between-class S_B and within-class scatter matrix S_W).
3) We compute the eigenvectors and corresponding eigenvalues for the scatter matrices.
4) For classifying we sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues and Use this d × k eigenvector matrix to transform the samples onto the new subspace.

Finally, this function returns a list containing

1) **mean**: Vector containing the mean of the columns of the data
2) **P**: Matrix containing the eigenvectors of the appropriate matrix
3) **D**: vector containing the variance explained by each Fisher discriminant dimension

BC - Building a k-NN face classifier via the eigenfaces approach

Preliminaries

When reading an image using the `OpenImageR` package, it is stored as a 3D array containing matrices for each of the color channels of our images of number of rows and

columns according to the images' height and width. In order to apply PCA, we vectorize the images so we can describe a whole image as one row in our matrix.
The database we get is composed of 150 RGB images of 180*200 pixels; therefore, each observation is placed as a vector in a space of 108,000 dimensions.
The aim of our work is to identify a specific person in the database with an image. So, even though an image's filename contains more information, we are going to only utilize the first number which can be considered as the person's ID. We'll store our labels in a character vector for later use.
Note that the classifier will use all the images we have as the training data. We will disclose why we choose certain parameters later in this document.

Fisherfaces

Using PCA and the Eigenfaces [3] method, we first project the data into the eigenfaces' space. Next, we run our `fda` function on the training data, getting a list containing the mean of its columns, the 24 eigenvectors, due to the fact that the model is classifying 25 classes, and the proportion of variance each linear discriminant represents. We then project that data onto the eigenspace given by the eigenvector matrix after using FDA.

k-Nearest Neighbors

When using k-NN we have to compute the distance of the observation that has to be labeled to that of all of the data points in our training set. We then sort `k` of those points based on the distance. The unlabeled observation will be labeled as the label that appears most frequently on the list we selected. In case of a tie, we have chosen to set the label as the one of those labels tied closest to the unlabeled one.
We use our projected data to compute the distances to every point and pick the distances of our image with the training data using the `euclidean` distance method. We sort the observations and use the rules aforementioned to select a label. We have just assigned our unlabeled image a label.

Validation Strategies

We used validation strategies to optimize certain parameters of our model:

- How much variance we are retaining
- The number of k-nearest neighbours
- The distance method used
- The threshold to determine when an image belongs to the database

In order to do this, we broke down our code into smaller functions:

`findK(obs, train.data, method, maxK)`

Using a projected image and projected training data, including labels, it runs the knn algorithm using the method specified for every `k` from 1 up to `maxK`. We decided on using a `maxK` value because it did not add much computational complexity to the program. This function returns a vector containing the label predicted for each of the k's tested whose last

two elements are the real label of the test observation and the farthest observation which has the same label as the test..

`testMethod(test.data, train.data, method, maxK)`

Given projected and labeled test and training data, it runs `findK` on all elements of the test set using `method` and `maxK`. It returns the accuracy for each k on each element of the test set as a vector and the maximum distance of those obtained using `findK`.

`plotGenerator(th, maxK)`

This function generates a bar plot for a specific threshold `th` of percentage of variance that displays the accuracy of the classifier for each `k` and the three distance methods we were recommended to try, `manhattan`, `cosine` and `euclidean`.

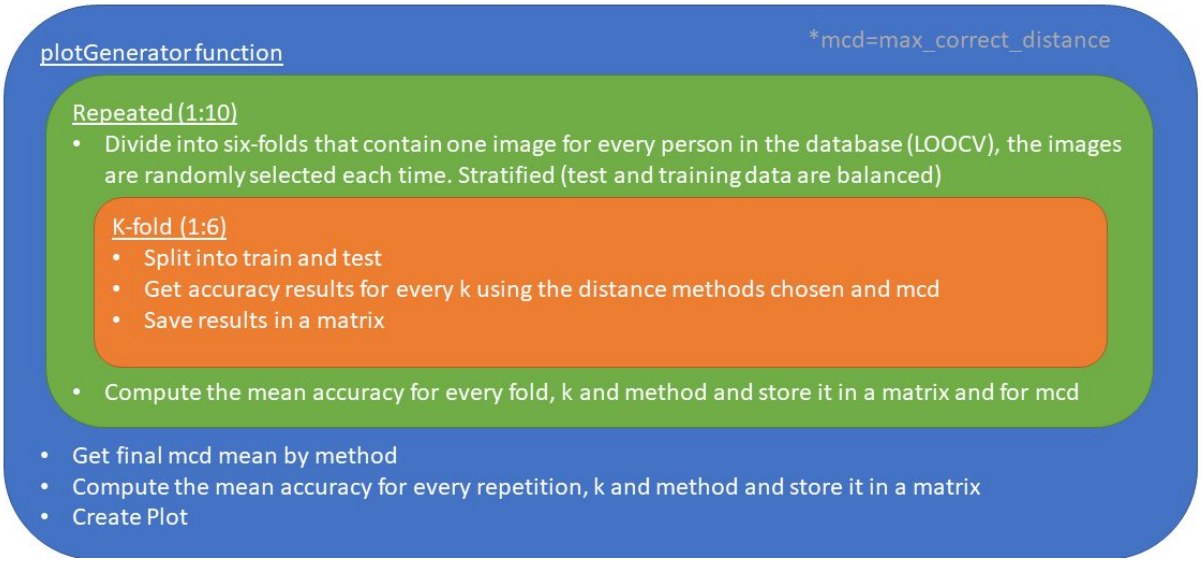


Figure 1: Diagram of the plotGenerator function

Repeated stratified k-fold cross-validation

Inside the `plotGenerator` function is where we implemented our validation strategies. We first added a column with the index of our dataset so when we used `str` from the `splitstackshape` package we just stratify 2 columns which is less computationally expensive than using the full 108,000. We use `str` inside our loop for the repetitions and end-up with 6 folds of data, containing each 25 images from different persons. We loop through them calling `testMethod()` and obtain the data necessary for creating the accuracy plots.

Optimization Results

After running the functions we created, we get the following plots:

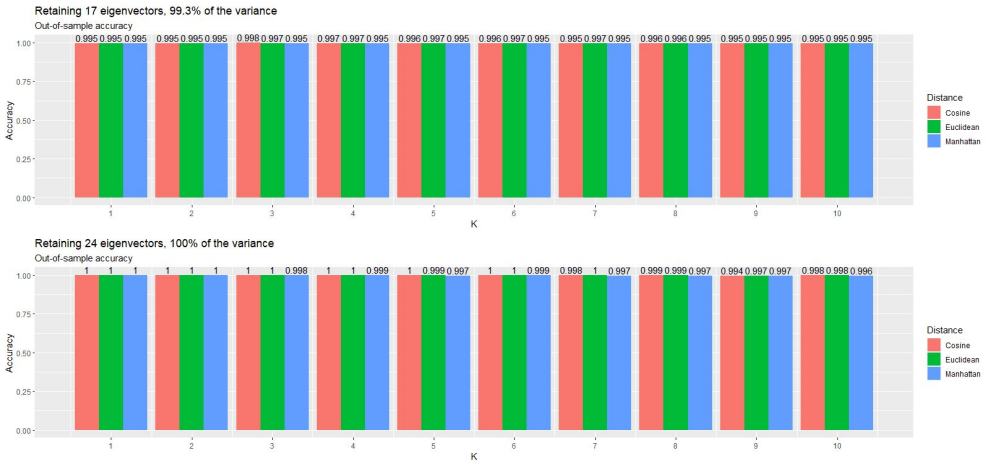


Figure 2: *generatePlot results*

As we can see, we are achieving almost 100% accuracy on all of our tests, which reassure us of the high quality of this model and the Fisherfaces technique. Through other plots retaining different percentages of variance, we discovered that the euclidean method was the most stable among all of them, the highest mean accuracy. In addition, we also saw that k=4 was the most reliable as its accuracy changed little from plot to plot.
We decided on keeping 17 eigenvectors because we think extracting the most relevant features and leaving behind little details will avoid overfitting our predictor with the training data. In addition, the runtime difference was a negligible value of 0.01 s.
We got the threshold for an image to be in a database retaining 17 eigenvectors to be 1598.28, obtained through the plotGenerator function.

Further Improvement

When we started thinking about stratified subsampling we thought that maybe the length of hair would be a good idea so we decided to categorize each person according to their hair length (if it was long enough so that the bottom of the face wouldn't be in contact with the green background). We determined it wasn't a great idea and left it there. But playing with plots, we found that it was somewhat relevant. Using the first 3 linear discriminant dimensions we obtained:

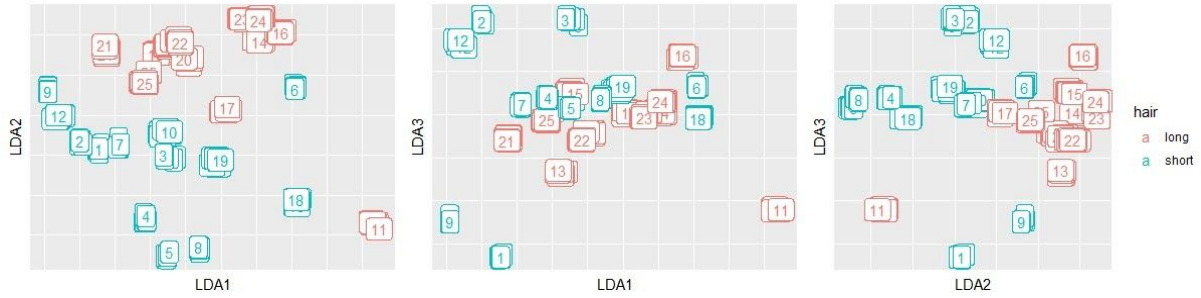


Figure 3: *·3 LDA and labeled hair data plots*

These 3 dimensions contain 68% of the data. We observed that the pictures where people have long hair tend to cluster together, as seen by the stacking of the labels. Therefore, we hypothesize that the model would benefit from having one model for each type of hair length. This approach will also bring consequences that derive in higher complexity maintaining the face recognition system, since there's a need for two different databases and having to track every user's hairstyle changes.

Conclusion

From this assignment In a few words,we can say that with Fisher we obtained higher accuracy but we can't be sure that it's always going to be the case: In fact we used PCA first to reduce our data which helped a lot in reducing the computational time and the costs. We can say that the PCA is an unsupervised algorithm that attempts to find the orthogonal component axes of maximum variance in a dataset , while the goal of LDA as supervised algorithm is to find the feature subspace that optimizes class separability and both can be of a great use combined together.

References

[1] T. Klosowski, "Facial Recognition Is Everywhere. Here's What We Can Do About It.," *Wirecutter: Reviews for the Real World*, Jul. 15, 2020. https://www.nytimes.com/wirecutter/blog/how-facial-recognition-works/ (accessed Nov. 01, 2020).

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, Jul. 1997, doi: 10.1109/34.598228.

[3] "Eigenface," *Wikipedia*. Oct. 01, 2020, Accessed: Nov. 01, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Eigenface&oldid=981330287.