Tarea 1 - Face Recognition, PCA and classification

CC5509: Reconocimiento de Patrones

Submitted To:
Mauricio Cerda Villablanca

Submitted By :
Alexandre Poupeau

May 6, 2019

Contents

1 Resumen **2**

2 Introduction **2**

3 Experiments and implementation **2**
3.1 Classification on raw data 2
3.2 Classification after using PCA 4
3.2.1 PCA and its impact 5
3.2.2 Classification with kNN 7
3.2.3 Classification with a Multilayer-Perceptron 9

4 Conclusion **10**

1 Resumen

This report focuses on the classification of different faces. It mainly studies the impact of light illumination on faces to classify and the benefits of dimensionality reduction on images before classifying. It also compares the performance of the kNN and MLP models for face classification.

2 Introduction

We used the YaleB+ dataset (DownloadData / MoreInformation) to make the experiments. The face images have a width of 168 and a height of 192. We only used 28 different persons (from B11 to B39, there is no B15), so there were only 28 classes. We only used frontal faces. However we used 64 different types of light illumination described with a azimuth and elevation value. A positive azimuth implies that the light source was to the right of the subject while negative means it was to the left. Positive elevation implies above the horizon, while negative implies below the horizon.

The training set was composed of the 7 images with the highest illumination strength for each person. The illumination strength (also called IST) is a metric that we created to measure the light illumination importance on the face. It is expressed as follows for a given image : $IST = |azimuth| + |elevation|$. Beware, in this sense, the bigger the illumination strength the darker the image (this is counter intuitive but we worked with this definition). Thus, the training set has a shape of $(28 * 7, 168 * 192)$ and the testing set $(28 * 57, 168 * 192)$ (rows, columns).

Now that it is more clear what we are working with, we are going to present briefly the different experiments. First, we classified faces using the k-nearest neighbours (kNN) algorithm using the raw data, which means without any sort of preprocessing and then using dimensionally reduced data with principal components analysis (PCA). We also studied in depth how PCA affects the result and what is the impact of PCA on images. Finally, we used a multilayer perceptron (MLP) to classify and see if it performs better than kNN.

3 Experiments and implementation

In this part, we will explore the experiments and how we implemented it in Python.

3.1 Classification on raw data

First, we tried to classify with kNN on raw data. We wanted to know what is the best number of k number of neighbours. Therefore, we just developed a simple loop and at the end plotted the accuracy as a function of k :

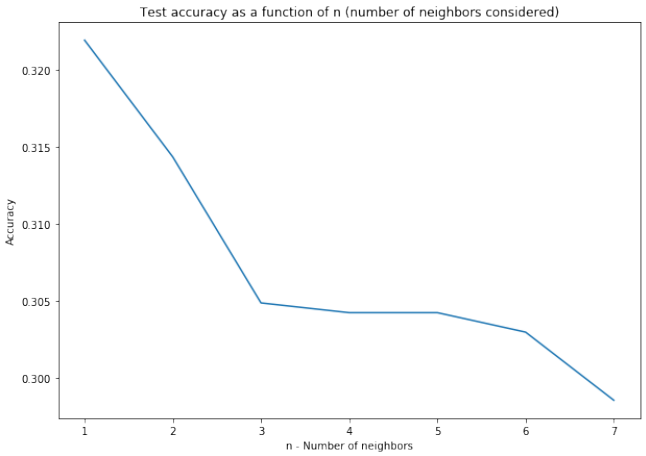


Figure 1: Test accuracy as a function of the number of neighbours used

This graph tells us a lot of information. On one hand, obviously the best number of neighbours is 1. On the other hand, we can deduce that the data is messy in its dimension and hard to classify (it is easier to visualize it as we can not draw lines to separate classes). The best accuracy we can obtain is $\sim 32\%$ which is low.

Seeing the previous graph, we could just stop here and conclude that kNN with raw data is not efficient when it comes to classify faces. Nonetheless, it is a little bit more complex than that and we will see it right now in the very next experiment.

We then wanted to know the accuracy as a function of illumination strength. In order to achieve that, we reorganized the testing set given the strength illumination of each test image. Then we just had to iterate on every subset of images which illumination strength belongs to a particular interval (and obviously iterate increasingly over the intervals). We chose a step of 20 for the illumination strength because it divides well the data (remember that the bigger the illumination strength the darker the image):

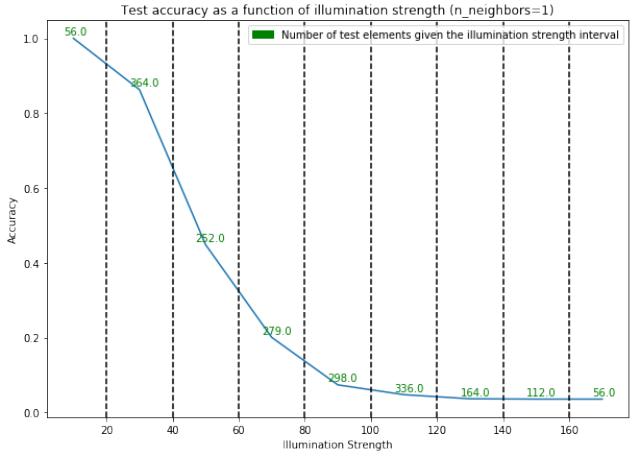


Figure 2: Test accuracy as a function of the illumination strength

The sum of the number of test elements is equal to $28 * 57$. What is striking is that finally the accuracy depends a lot on the illumination. For example, if the illumination strength is between 0 and 40, the accuracy is greater than $\sim 90\%$, and this result was obtained using more than 400 test images so it is relevant. This was quite expected since a dark face or only a half-illuminated face are both hard to classify using only fully illuminated faces as training data. However, the difference of accuracy was not expected to be that strong (when the the illumination strength is greater than 80, thus the images are really dark, and the accuracy is close to 0%).

We can now take a look at the confusion matrix obtained when using 1 neighbours on all the testing set and understand its structure better:

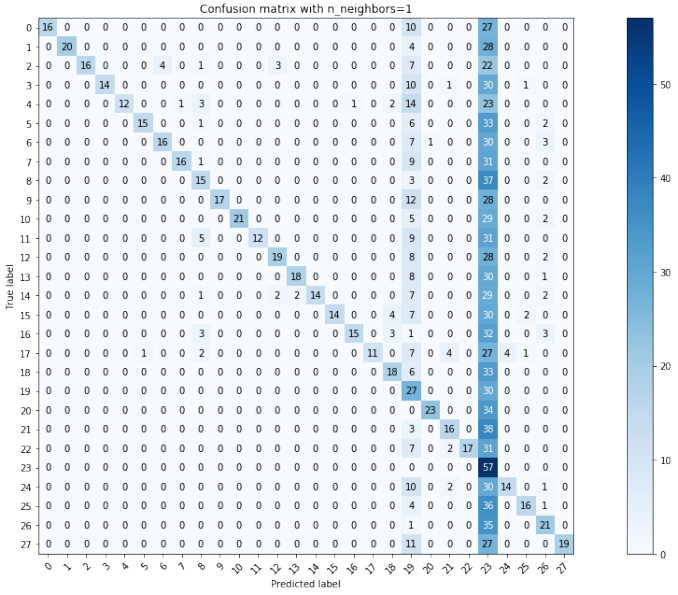


Figure 3: Confusion matrix using kNN with k=1

The confusion matrix is really bad but not messy. All the predictions lie on the diagonal or on two columns. How can we describe this phenomena ? This can be explained by the previous experiment. All the predictions on the diagonal are mainly from the test images which illumination strength is between 0 and 40. The rest of the predictions are in two columns because the persons 19 and especially the 23 may be darker in the training set. Then the kNN algorithm simply associates a darker test images with the class (or person) 19 or 23.

3.2 Classification after using PCA

In this part, we precise that because of computation problems, we had to work with images of size $84 * 96$ instead of $168 * 192$ and therefore recreate our dataset. Thus, the training set has now a shape of $(28 * 7, 84 * 96)$ and the testing set $(28 * 57, 84 * 96)$ (rows, columns).

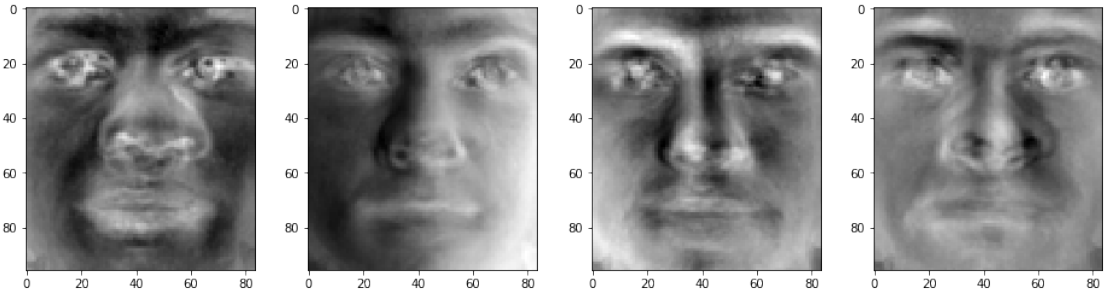
3.2.1 PCA and its impact

We created a function $myPCA(data = train\_data, n\_components = k)$. It is simply a function that returns the mean image from the train data matrix X and the best k principal components. First, it computes $\bar{X} = X - \mu_X$. Then it sorts the highest values in the s matrix (eigenvalues $= s_{i,i}^2$) and keeps the indexes. As VT matrix contains the eigenvectors of the matrix $\bar{X}^T\bar{X}$, we took the k best eigenvectors from VT matrix rows using those indexes. The parallel with PCA is that if X is centered (this why we compute $\bar{X}$) then the covariance matrix $C = \frac{X^T X}{n-1}$ (n =number of training samples) and therefore we in reality extract the eigenvectors from the covariance matrix. Here is the code :

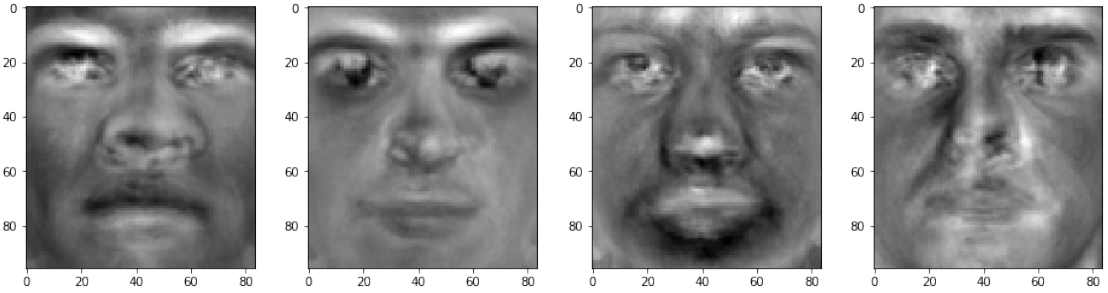
```
def myPCA(data, n_components):
    """
        Arguments:
            data (array) : shape (N_samples, n_features)
            n_components (int) : numbers of first principal components to keep

        Returns:
            best_n_eigenvectors (array) : has a shape (n_features, n_components)
            mean_img (array) : has a shape
    """
    # mean image
    mean_img = np.mean(data, axis=0)

    # center the data for input svd
    input_svd = data - mean_img

    # Singular-value decomposition
    U, s, VT = np.linalg.svd(input_svd, full_matrices=False)

    # get the n_components highest eigenvalues indexes
    ordered_n_best_eigenvalues_idx = np.zeros(n_components)
    for i in range(0, n_components):
        index_max = np.argmax(s)
        ordered_n_best_eigenvalues_idx[i] = index_max
        s[index_max] = 0

    # get the best eigenvectors
    best_n_eigenvectors = list()
    for idx in ordered_n_best_eigenvalues_idx:
        best_n_eigenvectors.append(VT[int(idx), :])
    # here now best_n_eigenvectors has a shape of (img_size, n_components)
    best_n_eigenvectors = np.asarray(best_n_eigenvectors).T

    return best_n_eigenvectors, mean_img
```

In order to visualize the best eigenvectors as images we had to make a transformation in the data. If we have a vector which coordinates are in $[a, b]$, if we want it to be in $[\alpha, \beta]$, we need to apply

the following function $f(x) = \frac{x-a}{b-a} \times \beta + \alpha$ to every coordinate. For each eigenvector e, we chose $a = min(e)$, $b = max(e)$, $\alpha = 0$ and $\beta = 255$. This way we could visualize what the 8 best eigenvectors look like (we chose to display only 8 and not 10 because of space and because they do not bring more particular information) :



(a) PC1 (b) PC2 (e) PC5 (f) PC6



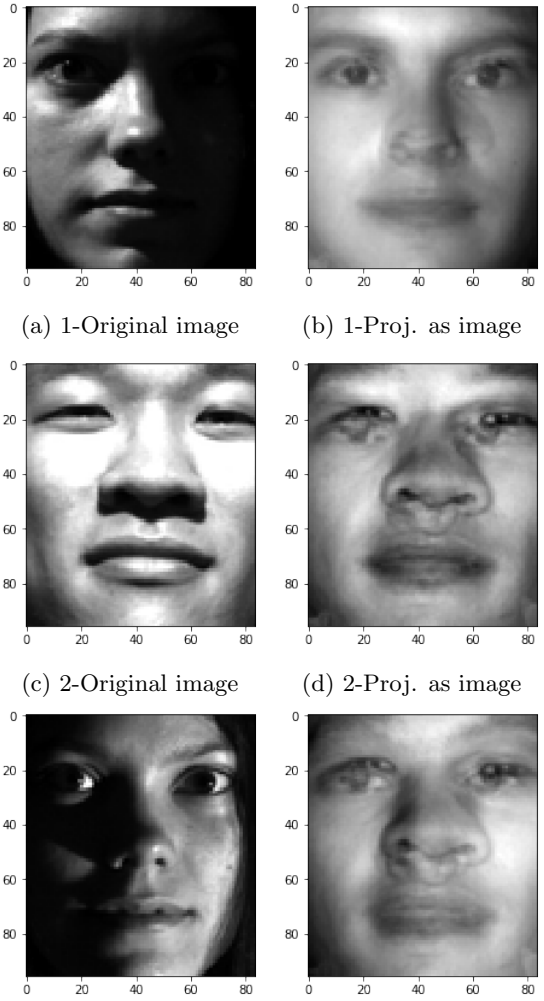
(c) PC3 (d) PC4 (g) PC7 (h) PC8

It is interesting to notice that the first principal components describe different types of faces and focus on different characteristics or features (eyes, nose, eyebrows, mouth ...). From PC6, images start to be messy and do not describe more special characteristics.

We then created a function named $getProjection(Img_{vec}, train\_data, k)$. This function only calls the myPCA function, and compute $Img_{vec} \times M$ where M is the matrix with the k best eigenvectors in columns. The function simply returns a vector of k dimension. Here is the code :

```
def getProjection(I, train_data, k):
    """
        Compute projection of the image using the k first principal components
    """
    k_eigenvectors, mean_img = myPCA(data=train_data, n_components=k)
    return np.matmul(I, k_eigenvectors)
```

We obtained the projections as images computing $Img_{vec} \times MM^T + mean\_img$ (note that if we use all eigenvectors $MM^T = I$ identity, thus $Img_{vec} \times MM^T + mean\_img = Img_{vec} + mean\_img$). Here are three test person faces before and after applying projection with $k = 3$ (three persons instead of five to not overload the report):



(a) 1-Original image (b) 1-Proj. as image

(c) 2-Original image (d) 2-Proj. as image

(e) 3-Original image (f) 3-Proj. as image

(g) Comparison original image and projection as image for three persons in the test dataset

What is amazing is that even with only $k = 3$ components, we are able to recognize the two first persons very easily. Therefore, an algorithm such as kNN may be able to classify faces as well with the projections. We chose on purpose the two first persons because the result is impressive. It was to show how powerful PCA can be. Nevertheless, in general it is much less impressive. We can see it with the third image which is the same person as the first one but here its projection as image does not look like the original.

3.2.2 Classification with kNN

We wanted to know, given k the number of eigenvectors used for projection, the best number of n neighbours. Thus we simply made two loops, one for n and one for k and compute the accuracy

for each combination. Finally we can get this graph :

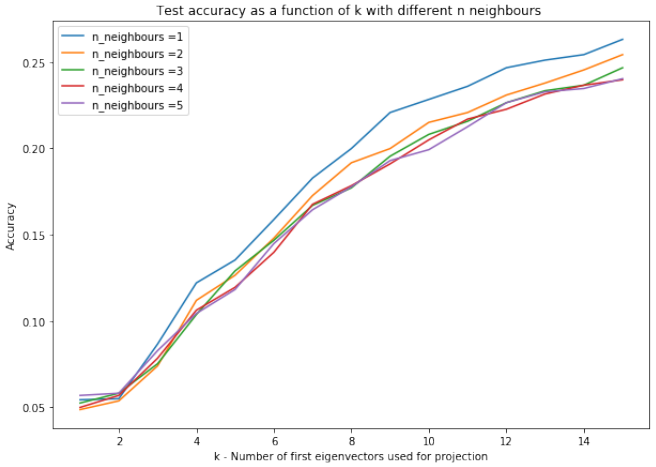


Figure 6: Accuracy as a function of the number of the k first eigenvectors used for projection for different n neighbours

As we can see the accuracy grows as we use more principal components which makes total sense. Moreover, the best number of n neighbours is still always 1, as we found with raw data. Hence we can conclude that the optimal number of neighbours does not have any correlation with the number of k best components used for projection.

We now want to know the accuracy as a function of the illumination strength and given different values of k (number of best eigenvectors used) :

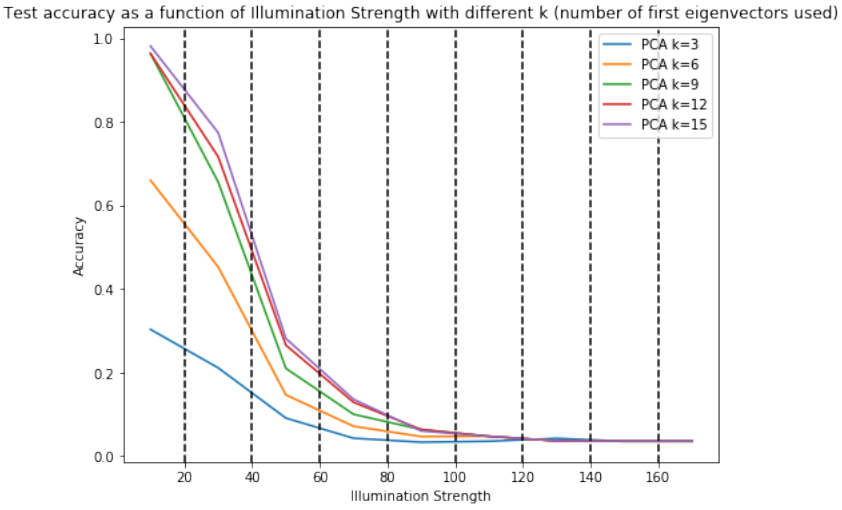


Figure 7: Accuracy as a function of the illumination strength with kNN for different k

What is interesting is that from $k = 9$ we almost have the same curve as with the raw data which

is great. This shows that even if we reduce the data with a feature size of $168 * 192 = 32256$ to 9, we still almost get the same result. Almost because with the raw data we got $\sim 90\%$ accuracy in the interval $[20, 40]$, here we just have $\sim 80\%$ in this exact same interval). We can conclude that PCA is really powerful in this case and allows us to reduce considerably our feature dimension without losing to much information.

3.2.3 Classification with a Multilayer-Perceptron

We wanted to see if we could obtain better results with a MLP. We used the sklearn library in order to train a MLP model. We managed to get a good model by testing with different hyperparameters such as the number of layers, the number of neurons in each, the activation functions etc... Here is the accuracy as a function of the illumination strength :

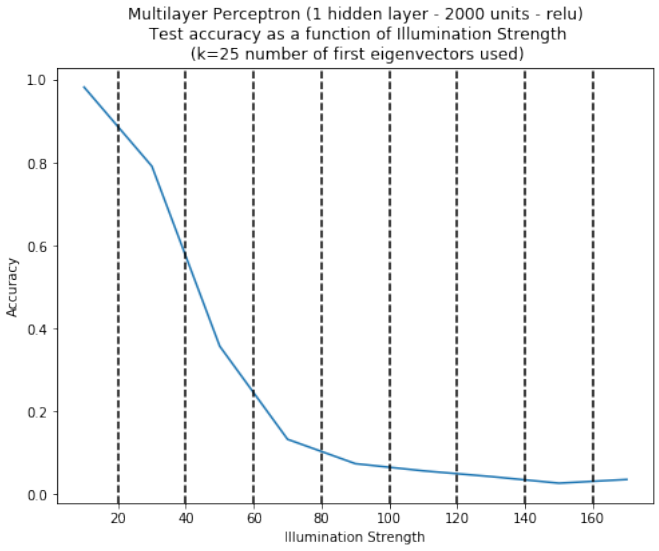


Figure 8: Accuracy as a function of the illumination strength with a MLP (k best components used =25)

Finally this model is not extraordinary and is just as good as kNN in this case. Moreover we had to test a lot of configuration to get this result. Plus, we had to use up to 25 principal components to get this curve whereas with the kNN we get these result with only around 10 principal components.

You could notice that our MLP model is not really deep : there is only one hidden layer. What was funny to observe during the hyperparameters tests, is that, in that particular case of face classification after applying PCA, what was working best was only one hidden layer with lots of neurons. This was quite unexpected, we started to try with several smaller layers.

4 Conclusion

To conclude, in order to classify person faces, the light illumination on the face has a really strong importance. Classifying is a possible task and we can get good results and accuracy if we try to predict on images that are well-illuminated.

PCA is really impressive for dimension reduction in that case, and allows us to shrink the huge feature dimension from the number of pixels in the image (32256) to only around 10 dimensions without losing that much information. However, it does not help much to counter the illumination problem. Predictions using raw or dimensionally reduced data will be overall the same given the same illumination strength.

Finally, the MLP model does not perform particularly better than kNN in that case which does not make it a preferable choice over the kNN, which is far more simple thus preferable, for using it to predict person faces.