Facial Recognition with Eigenfaces and Fisherfaces

Anya Jensen and Noah D'Souza

September 26, 2018

Abstract

This report details the concepts, math, and implementation of eigenfaces and fisherfaces for facial recognition.

We begin with eigenfaces, which uses principal component analysis (PCA) to create a set of images that can be put together in varying amounts to create a face, and then match that face to an existing image of a face.

Next, we use fisherfaces for facial recognition. Fisherfaces implements linear discriminant analysis (LDA), which takes differences between pictures of various people into account more than it takes into account the differences between varying pictures of the same person. This theoretically could make the facial recognition algorithm more accurate.

We compare the performance of both algorithms to see which is best equipped to run facial recognition on a pre-prepared database of images of people, wherein there is more than one image of each person within the database. The algorithms will be assessed based on their accuracy in matching faces and their computational robustness and speed.

Introduction

Facial recognition is the process of matching a picture of one face to a different picture of the same face. It has various applications in fields such as security and criminal justice.

We will use two related methods of facial recognition, eigenfaces and fisherfaces, and implement a program than can identify and match a person's face with another picture of them, or at the very least the most mathematically similar image in the programâĂŹs set. We will discuss which algorithm is more effective, and in which scenarios a specific algorithm is more useful.

To start, we will implement eigenfaces by explaining the math behind singular value decomposition (SVD) and PCA. We will then move on to fisherfaces and explain how much it is related to eigenfaces mathematically, but differs and builds upon it in key forms (such as the introduction of LDA).

Eigenfaces

1.1 Introduction

A picture of a face can be represented as a combination of eigenfaces. For example, a person's face can be made up of 10% of eigenface 1, 7% of eigenface 2, etc. in a set of eigenfaces. An eigenface is generated by executing Principal Component Analysis (PCA) on a large set of training images, which we will discuss in the following section.

To develop facial recognition through eigenfaces, begin by acquiring a set of images (preferably in the same lighting conditions and face positions). Set aside some amount of these to be used as a testing set, and the remaining M images will be used as training data.

1.2 Computation

In this implementation, all the of the images are nxn pixels in size (our images were 256x256 pixels). Now, transform each image into a column vector, and then place the column vectors into an n^2xm sized matrix of all the images. This will simplify much of the future matrix computations involved in PCA. Let each column vector be called Γ_i, where i is the index of the image as a column vector. Do the same with the test images in a separate matrix, I.

Start by finding the average Ψ of the training set

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$$

An average face can be seen in figure 1a.

where M is the number of images in the training set, and Γ is the images in the training set. This will allow us to mean-center our images using the equation

$$\Phi_i = \Gamma_i - \Psi_i$$

Mean-centering the images reduces unnecessary variation.

The new mean-centered data can be used to find the covariance matrix C of the set, which is defined as

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T$$

The covariance matrix represents the variation of every data point from the mean of the set in n^2-dimensional space. In addition, the set of eigenvectors U of this covariance matrix produces our set of eigenfaces. You can see an image of an eigenface in figure 1b.

Now that the eigenfaces have been isolated from the covariance matrix, we must find the weights of the eigenfaces within each image

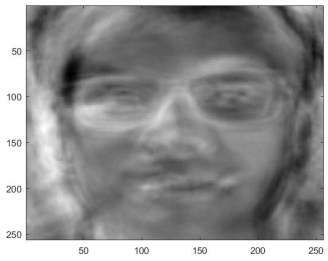

Figure 1a

This image is the average face from a set of faces.As you can see, many of the features are blurred together, and many features are just overlapping (i.e. people smiling and not smiling and multiple sets of glasses appearing on the average).

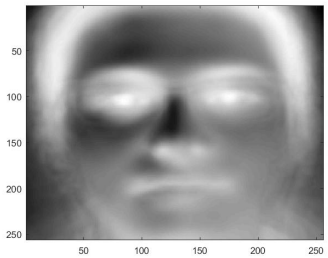

Figure 1b

This image shows an eigenface from our training images. It has very undefined features, but appears to be much more cohesive than the average face in figure 1a.

by separately multiplying the transpose of the eigenfaces by the mean-centered image vectors and by the test image vectors I

$$\Omega_{training} = U^T \Phi$$

$$\Omega_{test} = U^T I$$

Where $\Omega_{training}$ is the weight for the training set and Ω_{test} is the weight for the test set.

The weights represents how much of each eigenface is needed to form an image of a specific face. Note: not all of the eigenfaces are needed to produce accurate results, as some of them (usually the first ones in the set) are more "important" than ones later in the set. This implementation only used the first 40 eigenfaces.

The resulting matrices $\Omega_{training}$ and Ω_{test} contain eigenface point representations of the images in eigenspace, which is essentially a compressed, eigenvector based version of the in n^2-dimensional space from previous sections. This entire process is known as principal component analysis.

1.3 Recognition

Mathematically speaking, points in eigenspace that represent different images of the same people should be close together relative to images of other people, therefore to match a face from the test set to and image in the training set, all one has to do is find the nearest training image point to the test image's point in eigenspace. This is done by taking the euclidean distance between the test image's point and all the training image points

$$\epsilon = \sqrt{\sum_{n=1}^{M} \left(\Omega_{test_n} - \Omega_{training}\right)^2}$$

The training image point corresponding to the smallest euclidean distance is what the eigenface program believes to be a match.

Fisherfaces

2.1 Introduction

Fisherfaces also implements eigenfaces, but it uses linear discriminant analysis, or LDA. The main difference between fisherfaces and eigenfaces is that fisherfaces implements classes. In this context, classes are defined as a set of different pictures of the same person. If the whole dataset has 100 images, and 20 pictures of each person, then there would be five classes. This distinction of classes allows to put more emphasis on the differences between individual people's

faces and less emphasis on the differences between images of the same person.

2.2 Computation

Fisherfaces implements many of the same steps as eigenfaces, and mostly just expands on it as we explain later. For now, follow the same steps at eigenfaces, up until you have your eigenfaces. Now we will move onto the expansion fisherfaces has on eigenfaces.

In order to collapse variation within classes and expand variation between classes, we must find the scatter of data within classes, (S_b), and the scatter of data between classes, (S_w) such that:

$$S_b = \sum_{i=1}^{C} N(\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$$

$$S_w = \sum_{i=1}^{C} \sum_{X_k \epsilon X_i}^{M} (U - \mu_2)^T(U - \mu_2)$$

N would be the number samples in class X_i, X_k would represent the kth class in the set, and μ_i would be the mean image of class X_i while μ would be the mean of all images. These two equations reduce the differences inside classes so that point representations of images of the same people are closer together in fisherspace (which is like eigenspace, but with LDA factored in), and the point representations of images of different people are farther apart in fisherspace.

The set of eigenvectors W_i of S_b and S_w is the set of fisherfaces such that

$$S_b W_i = \lambda_i S_w W_i$$

with λ_i being the eigenvalues. This set of eigenvectors is used to find the weights of each face (in a similar manner to the eigenfaces algorithm) with the equations

$$F_{train} = W_i^T U^T \Phi$$

$$F_{test} = W_i^T U^T I$$

which define F_{train} and F_{test}. These two are the fisher equivalent of Ω_1 and Ω_2 from eigenfaces, but this time in fisherspace. You can see a fisherface in figure 2a. This process is essentially PCA with the addition of classes, also known as linear discriminant analysis.

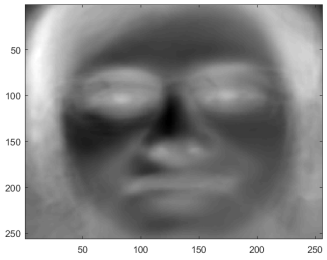

Figure 2

This image is a fisherface from our training set. It appears to have more defined features than those of the eigenfaces. This image is used in some amount to make full images in the dataset.

2.3 Recognition

The process of actually matching faces with fisherfaces is nearly identical to that of eigenfaces. It once again involves taking the euclidean distance between the test image and all training images, and

finding that the smallest distance begets a match, or at least the most similar image such that

$$\epsilon = \sqrt{\sum_{n=1}^{M} \left(F_{test_n} - F_t rain\right)^2}$$

Comparisons

We tested eigenfaces using 66 training images and 66 test images. We tested fisherfaces using a different set of images; we used 129 training images and 43 test images.

We found that fisherfaces were more accurate than eigenfaces, correctly guessing 90.7% of our faces. eigenfaces were able to accurately detect 89.4% of the faces we tested. eigenfaces, however, did run much faster than fisherfaces, with a total runtime of .719 seconds compared to fasherfaces' 4.918 seconds runtime.

The accuracy of eigenfaces remains pretty steady at around 87% once you pass 10 training images. Fisherfaces, however is much more non-linear, and changes between an accuracy of around 95% and 78%. You can see this clearly in figure.

It is hard to make considerable conclusions from these statistics, as the fisherfaces have much more training images and test images, which could lead to the increased runtime.

Whether fisherfaces or eigenfaces is more useful depends on the context. Eigenfaces could be useful in searching for ID photos or mugshots in a database (especially if time is of the essence), as all the photos were taken in the same lighting. They would not be useful when searching for a face match given a shot from a security camera as the test image.

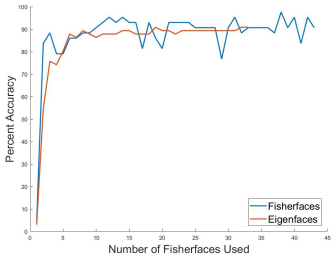

Figure 3

This graph shows the differences in accuracy between fisherfaces and eigenfaces. Both begin to plateau at around 10 training images; however, the accuracy for fisherfaces tends to vary, whereas the accuracy for eigenfaces tends to stay constant.

Conclusion

In conclusion, fisherfaces make more sense in real-world settings, where changes in lighting and position would make it necessary to collapse variation between pictures of the same people. However, when conditions are ideal, eigenfaces are a much faster method. Our next steps would be testing our algorithms in more realistic settings, where faces are blurry or turned at angles, and see how our algorithms perform.

References

Belhumeur, Peter N, et al. *Eigenfaces vs Fisherfaces: Recognition Using Class Specific Linear Projection*. 7th ed., vol. 19, 1997, pp. 711-720

Pentland, Alex, and Matthew Turk. *Eigenfaces for Recognition*. 1st ed., vol. 3, Journal of Cognitive Neuroscience, 1991, pp. 72-86

Eigenface. Wikipedia, Wikimedia Foundation, 3 Apr. 2018, en.wikipedia.org/wiki/Eigenface