Facial Recognition

Kyle Combes and Paul Nadan

March 26, 2017

Abstract

In this module, our goal was to build facial recognition software capable of recognizing a photo of our classmates. Input images were pre-cropped and converted to grayscale and backgrounds were a consistent solid white. We initially implemented two approaches, eigenfaces and an artificial neural network. The eigenface approach was 100% and 82.5% effective respectively on two different test data sets, while the neural network approach was only able to reach accuracies around 15% due to a lack of sufficient training data. We then switched to a linear regression approach, which yielded accuracies of 100% and 95%.

1 Introduction

Facial recognition is the process of classifying an image of a human face as the face of a particular person. Images are typically converted to grayscale ("black and white"), which means that each pixel in the image is the brightness value, or "intensity," of that pixel. Since a grayscale image (as a computer sees it) is just a two-dimensional array of pixel intensities, it can be represented as a matrix, where each element corresponds to one pixel. Once you're looking at a matrix, you can use linear algebra to perform various operations on this matrix. Figuring out what operations to perform on this matrix is the challenge that beholds those attempting to classify the images.

Most facial recognition algorithms involve machine learning, as the algorithm learns which features identify a person's face from a set of training data. However, it is possible for algorithms to over-fit themselves to the training data, such that they can perfectly recall what they have learned but not generalize to new examples. As a result, cross-validation is used, where-in a different set of data is used to test the algorithm after it has been trained.

2 Algorithms and Justification

Numerous approaches and algorithms have been proposed to solve the problem of facial recognition. Each one presents its own strengths and weaknesses. Some approaches are computationally cheap (i.e. they run quickly) to train and moderately accurate, while others are more computationally expensive to train and require more training examples but provide higher accuracy.

2.1 Eigenfaces

The eigenface approach is one of the faster, less accurate approaches to facial recognition. It requires only a couple photos of a person, but isn't very resilient to changes in lighting and face angle relative to the camera.

The idea behind it is that, when looking at lots of photos of faces, certain "features" of these images vary more and can be used to distinguish between different faces. Now, "features" in this

sense doesn't necessarily mean eyes, noses, mouths, or anything that humans might think of as potentially defining a person. Rather, they are determined by looking at how pixels in the image vary from image to image. This requires the use of some linear algebra concepts to extract the features.

When dealing with m images of dimension p pixels by p pixels, the first thing we did was resize our $p \times p$ image matrices into column vectors of length p^2. Then we horizontally concatenated these vectors into a matrix A, where $A \in \mathbb{R}^{p^2 \times m}$.

Once we had all of our image data represented in a single matrix, we mean-centered the data by subtracting the mean of each row, μ_j, where j is the row number.

$$A = \begin{bmatrix} x_{11} - \mu_1 & x_{12} - \mu_1 & x_{13} - \mu_1 & \dots & x_{1m} - \mu_1 \\ x_{21} - \mu_2 & x_{22} - \mu_2 & x_{23} - \mu_2 & \dots & x_{2m} - \mu_2 \\ x_{31} - \mu_3 & x_{32} - \mu_3 & x_{33} - \mu_3 & \dots & x_{3m} - \mu_3 \\ \dots & \dots & \dots & \dots & \dots \\ x_{p^2 1} - \mu_{p^2} & x_{p^2 2} - \mu_{p^2} & x_{p^2 3} - \mu_{p^2} & \dots & x_{p^2 m} - \mu_{p^2} \end{bmatrix} \tag{1}$$

Once we had this, we wanted to determine the directions in which the data varied the most. To do that, we used singular value decomposition (SVD) to calculate the eigenvectors of the covariance matrix $\frac{1}{\sqrt{m}} AA^T$. The SVD formula is as follows:

$$A = U \Sigma V^\intercal \tag{2}$$

where U is a matrix whose columns are the eigenvectors of $AA^\intercal$, Σ is a diagonal matrix containing the eigenvalues sorted greatest to least, and V is a matrix whose columns are the eigenvectors of $A^\intercal A$. Since the eigenvalues were sorted greatest to least, that meant the eigenvectors were sorted with the vector pointing in the direction of the most variation as the first column. Each successive column was an orthonormal vector pointing in the direction of the next greatest variation. (These vectors and their associated eigenvalues are termed "principal components" because they represent the directions of the most variation in the data.)

Next, we extracted an arbitrary number of principal components, discarding the rest of the eigenvectors, and used that to project the original face images into the vector space defined by the eigenvectors. After some experimenting with different numbers of principal components to get the best classification performance, we settled on using the top 30 principal components. This made our calculation to convert an image into eigenspace as follows, where $U^{(30)}$ is the first 30 columns of the eigenvector matrix:

$$W = AU^{(30)} \tag{3}$$

W could also be thought of as a linear combination of the eigenvectors in $U^{(30)}$ which best represented the original data. Each element in W represented a weight on an element of the eigenvectors. To visualize the eigenspace, we multiplied the eigenvectors by a ones column vector of length 30 and resized the result into a $p \times p$ model. The result can be seen in Figure 1.

To visualize a particular face projected into eigenspace, we similarly multiplied its weights by the eigenvectors (to project it back into higher dimensional $\mathbb{R}^{p^2}$ space), reshaped the result into a $p \times p$ matrix, and displayed it (Figure 2).

To determine which images were most similar to a test image, we looked at which each training image to find the one that was closest in eigenspace. Mathematically, all we had to do was compare the Euclidean distance between the two weight vectors. Thus, our program compared a test image's weight to each of our training image weights, found the closest match, looked up the name in a list using the index of the matched image, and reported that as the person it was a photo of.

Figure 1: A visualization of our eigenvectors. This was achieved by multiplying our eigenvectors by a ones column vector of length 30.

Figure 2: A visualization of the weights for Adam Selker. It was generated via projection of the weights vector for that image back into higher dimensional vectorspace. This was done by multiplies the weights vector by the 30 eigennvectors

2.2 Artificial Neural Network

Another approach to facial recognition is to use an artificial neural network. Neural networks are meant to model the way a brain works, using a number of neurons or nodes connected together. Using training examples the network gradually adjusts itself until it can accurately achieve the given task. A neural network requires many more training examples than the eigenface approach

and is much more computationally expensive to train, but it can produce better results.

A neural network consists of several layers of nodes, each of which is a function of the output of the nodes in the previous layer. By adding more layers with different functionality, you can find weights to appropriately apply each function to each input. The network illustrate in Figure **??** has two hidden layers (Layer 2 and Layer 3). Layer 1 is the input layer and Layer 4 is the output layer.

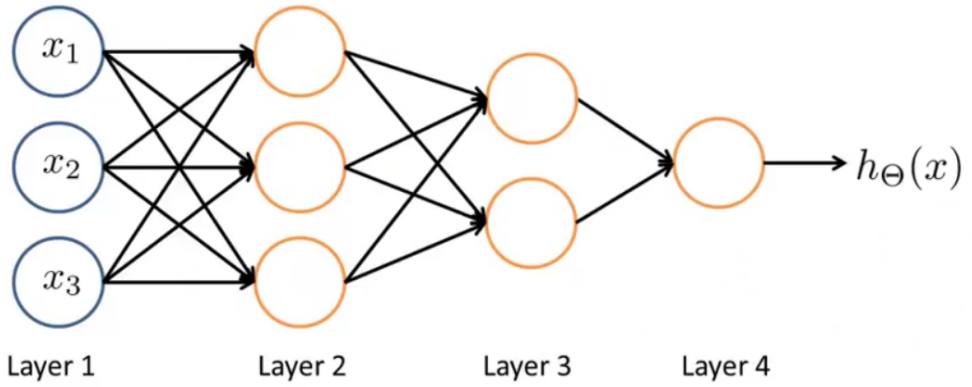

Figure 3: A neural network consists of various layers of "nodes," where each layer performs a specific function on the output from the previous layer. This model has two hidden layers (Layer 2 and Layer 3). Layer 1 is the input layer and Layer 4 is the output layer. Figure source: [1].

Our network uses two logistic regression layers, each of which multiplies the input by a weight and then passed the result through the sigmoid function (Equation 4), which scales input values to a number between zero and one. The input to the logistic function is a linear combination of the values of the nodes in the previous layer.

$$g(x) = \frac{1}{1 - e^{-x}} \tag{4}$$

Layer 1 contains the data being fed into the network, in our case the raw pixel values of the image. After passing through Layers 2 and 3, Layer 4 contains the confidence scores for that person in the class of matching the image.

To calculate each layer from the preceding one, a process known as forward propagation, we simply multiply the previous layer by a matrix of weights, Θ, and run the result through the logistic function (Equation 5). When computing the next layer, one additional node with a constant value is also added to the linear combination.

$$a_{n+1} = g(\Theta_n a_n) \tag{5}$$

Before using a neural network, it must be trained in a process called back propagation. Training data is fed into the network to compute the predicted values, which are then compared to the true results using a cost function (Equation 6). The weight values are then updated proportionally to the derivative of the cost function with respect to theta (Equation 6), which is known as gradient descent because the weights are changed in the decreasing direction of the gradient of the cost with respect to theta. The constant of proportionality, α, determines how rapidly the network tries to correct itself.

$$cost = J(\Theta) = \frac{-1}{m}(y^T log(y_{guess}) + (1 - y)^T log(1 - y_{guess})) \tag{6}$$

$$\Delta\Theta = -\alpha\frac{dJ}{d\Theta} \tag{7}$$

Computing the actual value of the gradient is more complicated, and involves finding the error of the nodes in each layer, starting at the last layer and working back, and then computing the gradient from the error values (Equation 8-11).

$$error_{n_{max}} = \delta_{n_{max}} = y_{guess} - y \tag{8}$$

$$\delta_{n-1} = \Theta_{n-1}^{T}\delta_n . * g'(\Theta_{n-1}a_{n-1}) \tag{9}$$

$$g'(\Theta_n a_n) = a_n . * (1 - a_n) \tag{10}$$

$$\frac{dJ}{d\Theta}(n) = \frac{1}{N_{faces}}\delta_{n+1}a_n^T \tag{11}$$

2.3 Linear Regression

Due to the lack of performance from our neural network approach, we decided to pivot the night before the due date. We settled upon using linear regression on our eigenface weights to classify images, rather than by finding the minimum Euclidean distance between images.

To do this, we first needed to calculate the linear regression weights. This required first prepending a constant term to our test image weights. More specifically, that meant prepending a column of ones to the weight matrix. Then we were able to use Equation 12 to find the weights B.

$$B = X^{-1}y \tag{12}$$

By multiplying the eigenvalues of test images by these weights, we were able to determine the probability of matching each face in the class (Equation 13).

$$y = XB \tag{13}$$

3 Performance

There are a number of ways to measure the performance of a facial recognition algorithm, such as the sensitivity (probability of identifying a positive result) and specificity (probability of identifying a negative result). However, we chose to use the accuracy, which is just the overall probability of a result being correct, because it was the clearest and most intuitive measure of success.

3.1 Eigenface Approach

Our eigenface approach was able to achieve 91.25% accuracy overall. More specifically, it was able to successfully identify 100% of the faces in the "easy" test set but only 82.5% of the images in the "hard" test set. (Each set consisted of 40 images.)

The eigenface approach required 1879 milliseconds to train itself, but was able to classify the 40 test images in 53 milliseconds.

3.2 Artificial Neural Network Approach

When implementing a neural network based on logistic regression, our prediction performance varied greatly depending on how we structured our network, but never reached a value above 15% accuracy on the easy set of test data. In an attempt to improve our results, we extended our neural network algorithm through two different means. On directly fed the eigenvector weights from the first approach into the network as input, and the other used convolution. Unfortunately, neither significantly increased the accuracy of the network. In the case of using the eigenvector weights, we were able to fit the training data well, but the performance did not extend to our test case. In the case of convolution, the network was able to learn the training data more accurately, but the test data accuracy dropped to around zero.

This failure most likely resulted from trying to use a model which did no feature extraction and simply performed logistic regression on the pixel intensities. Had we implemented a more complex model, such as one of the two proposed in [1], with 17 or 22 hidden layers performing various tasks, we could have achieved much better performance given sufficient training time and training data. Figure 4 shows the results of the neural network developed in [1] (OpenBR) versus eigenfaces on the Labeled Faces in the Wild (LFW) dataset.

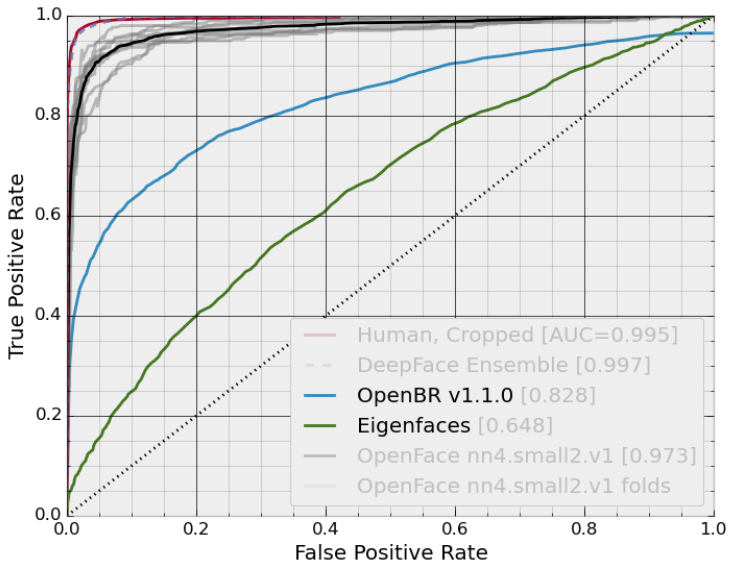

Figure 4: Had we been able to implement and train the neural network model proposed in [1] (OpenBR), we could have seen significant performance increases. However, we were unable to implement such a model in the time given. Figure source: [1].

Regardless, our rudimentary model took several minutes to train itself, depending on the number of iterations run, and classified images during the test phase in 676 milliseconds.

3.3 Linear Regression Approach

Our linear regression approach achieved a 97.5% accuracy overall, with 100% accuracy on the easy test set and 95% accuracy on the hard set. This accuracy is a significant improvement over the base eigenface approach.

Linear regression required 2010 milliseconds to train itself and 61 milliseconds to classify the 40 test images.

3.4 Comparison of Approaches

Overall, the linear regression approach proved to be the most effective for relatively small training sets. Although powerful, neural networks require huge amounts of data and processing power to reach their full potential, resources which we did not have for this project. Compared to the basic eigenfaces approach, linear regression proved to significantly improve performance for very little increase in processing required.

4 Conclusion

While there are many approaches to facial recognition out there, we chose to undertake eigenfaces, artificial neural networks, and a linear regression variation of eigenfaces. While the eigenfaces approach was fairly accurate for a small training set, the addition of linear regression offered significant improvement. Neural networks, however, proved to be ineffective for this scale of a project. Our next steps would be to implement a better model based on one of those in [1], acquire a much larger set of training data for the neural network, and allow it to run for a longer period of time, in the hopes of getting it to surpass the accuracy of the other approaches.

References

[1] F. Schroff, D. Kalenichenko, and J. Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clusetering*. IEEE Xplore, 2015.