The SVD and the Mathematics of Facial Recognition

Nicolas Bertagnolli

Everyday our brains are inundated with thousands upon thousands of images and many of them are filled with people and their faces. How do humans process and recognize faces? Do we look at parts of the face and their relationship to each other or are faces represented more abstractly in the human brain? These questions have troubled scientists for many years, and will most likely trouble them for many more to come. However, with modern mathematics and powerful computing capabilities we are beginning to shed light on these difficult question. The most successful method of facial recognition and classification to date is due to two brilliant scientists Mike Kirby and Lawrence Sirovich. They made use of eigenfaces to represent abstract patterns inherent in every human face. What is an eigenface you ask? Before we can get to that we need to develop a solid understanding of the singular value decomposition (SVD) and how it is used to extract meaningful patterns from noisy data.

The SVD is an incredibly powerful mathematical technique that was first roughly described and discovered by Beltrami and Jordan in 1873 while studying bilinear forms [**?**]. The SVD, as its name suggests, is a matrix decomposition that can be performed on any matrix. To motivate the initial problem let's try and find a simple yet meaningful way to diagonalize any matrix $A \in \mathbb{C}^{m\times n}$. This problem is more difficult and subtle than it might appear. The general method for diagonalization is:

$$A = S^{-1}\Lambda S \tag{1}$$

where S is a matrix of the eigenvevtors and Λ is diagonal with the eigenvalues on the diagonal. This particular diagonalization will only work for a square matrix, moreover; this diagonalization is not always possible, especially when the geometric multiplicity of the eigenvalues is less than the algebraic multiplicity. The SVD provides us a with a method for diagonalizing any matrix at the cost of two sets of "singular vectors" U and V instead of one S as in (1). The

SVD produces the following decomposition for any matrix A:

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

where U and V are unitary matrices, Σ is diagonal and positive, and * represents the conjugate transpose of a matrix. This remarkable diagonalization is achieved by loosening our restriction on the diagonalizing matrix S. Instead of one matrix that diagonalizes as in (1) we ask the question what happens if we allow our matrix to be diagonalized by two different matrices U and V? Now we can begin trying to construct our U and V. This "restriction loosening" isn't all bad, since we are constructing U and V for our own purposes let's choose them to be orthogonal. To better understand the power of this particular decomposition let's construct the SVD more formally and examine the corollaries and the three resultant matrices in more detail. The following proof is taken from Golub and Van Loans book on matrix computations and is one of the more concise formal constructions of the SVD [**?**].

Theorem

If $A \in \mathbb{R}^{m\times n}$ then there exists orthogonal matrices:

$$U = [\vec{u}_1 ... \vec{u}_m] \in \mathbb{R}^{m\times m} \qquad\qquad V = [\vec{v}_1 ... \vec{v}_n] \in \mathbb{R}^{n\times n}$$

such that:

$$U^T AV = \Sigma = \text{diag}(\sigma_1, ..., \sigma_r) \in \mathbb{R}^{m\times n}, \qquad\qquad r = \text{rank}(A).$$

where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$

Proof. Take $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ to be unit vectors, $\left\| \frac{x}{\|x\|_2} \right\|_2 = \left\| \frac{y}{\|y\|_2} \right\|_2 = 1$, such that they satisfy $Ax = \sigma y$ Where $\sigma = \|A\|_2$. Now by linear independence we know that there exists $V_2 \in \mathbb{R}^{n \times (n-1)}$ and $U_2 \in \mathbb{R}^{m \times (m-1)}$ such that $V = [x|V_2] \in \mathbb{R}^{n \times n}$ and $U = [y|U_2] \in \mathbb{R}^{m \times m}$ are orthogonal. This gives us:

$$U^T A V = \begin{bmatrix} \sigma & \omega^T \\ 0 & B \end{bmatrix} = A_1$$

where $\omega \in \mathbb{R}^{n-1}$ and $B \in \mathbb{R}^{(m-1)\times(n-1)}$. Since

$$\left\| A_1 \left(\begin{bmatrix} \sigma \\ \omega \end{bmatrix} \right) \right\|_2^2 \geq (\sigma^2 + \omega^T \omega)^2$$

we have that $\|A_1\|_2^2 \geq (\sigma^2 + \omega^T \omega)$ but $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$, which means that $\omega = 0$! The proof is complete with an induction argument. □

Now that we have a formal construction of the SVD let's look at some of the basic properties and see what we can learn. From (2) and the orthogonality condition that we recently imposed we get that, $AV = U\Sigma$ which can be visualized as:

$$A \begin{bmatrix} | & \cdots & | \\ v_1 & \cdots & v_r \\ | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | \\ u_1 & \cdots & u_r \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \tag{3}$$

This pictorial representation immediately sheds valuable light on the nature of U and V. First notice that each column $v_i \in V$ represents a member of the row space of A. This is because (3) gives us that $Av_i = u_i \sigma_i$. Since $\sigma > 0$ and u_i is an orthogonal vector we know that $\sigma_i u_i \neq 0$. This means that Av_i is not in the nullspace of A, and since the vectors comprising the null space of A form the orthogonal complement of those vectors comprising the row space v_i must be in the Row space! A similar argument demonstrates that the columns of U represent elements of

the column space of A! Already we are beginning to see structure emerge. How do we arrive at U and V? How does their calculation relate to our original matrix A? Well from (3) we see that:

$$\begin{aligned}
AV &= U\Sigma \\
A^TAV &= A^TU\Sigma && \text{Multiply by } A^T \text{ on both sides} \\
&= (U\Sigma V^T)^TU\Sigma && \text{Use the definition of the SVD} \\
&= V\Sigma^TU^TU\Sigma && \text{Apply the Transpose} \\
&= V\Sigma^TI\Sigma && U^TU \text{ is orthogonal} \\
&= V\Sigma^2V
\end{aligned}$$

This means that $A^TAv_i = \sigma_i^2v_i$. This is just the eigenvalue equation! This means that the columns of V are the eigenvectors of A^TA. A similar argument shows that the eigenvectors of AA^T are the columns of U. In only a few lines we have uncovered an intimate relationship between the eigensystems of A^TA, AA^T and A all with the help of this simple decomposition.

These properties are nice, but what is the real take away message from all of this? First of all, we can diagonalize any matrix using two separate matrices U and V, and these matrices provide the ideal basis for both the row space and the column space of A. Our algorithms will be based almost entirely on this fact so remember it.

Now that we have a basic understanding of the SVD let's apply it to the problem of facial recognition. The problem of facial classification is a difficult one. The first approaches used were rudimentary and not very robust. For example some of the first attempts at facial recognition consisted of looking at major facial features like the nose or mouth and their spacial relationship to one another [**?**]. Although at times these methods could be effective, they often broke down when the face was rotated or a different view of the face was presented to the algorithm. This led Kirby and Sirovich to propose the use of the SVD as a means to classify faces more effectively in their hallmark 1990 paper [**?**].

Sirovich and Kirby's idea was simple; that a face is a highly structured piece of data and can be represented accurately by a small set of patterns. Now the problem becomes how do we encode faces by this small set of patterns, and then use it to recognize a face? We begin by recognizing that an image of a face is just a matrix of pixel values call it A. This $A \in \mathbb{R}^{m \times n}$ pixel matrix representing a face is then unfolded column wise into a face vector $\vec{a} \in \mathbb{R}^{mn \times 1}$. These face vectors are then concatenated column wise into a new matrix $\Phi \in \mathbb{R}^{mn \times k}$ where k is the number of faces in the data set. This unfolding and concatenation can be visualized easily in Figure 1.

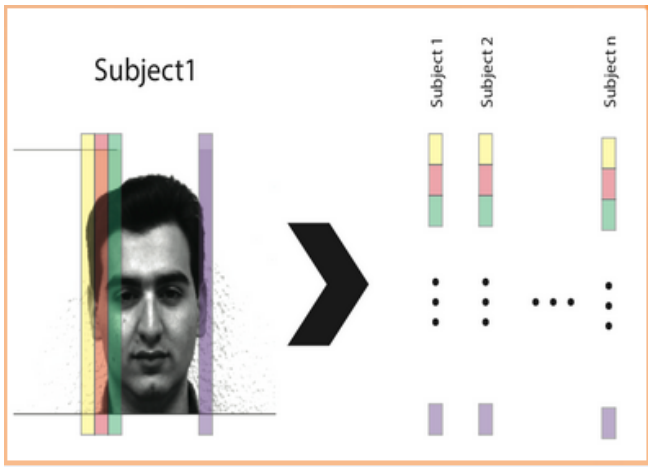

Figure 1: Visual representation of unfolding a face matrix into a face vector and recomposing these face vectors into a new matrix of faces[**?**].

Now we possess a matrix of faces that we can work with to uncover the hidden reduced dimensional face space. Each row of ϕ can be viewed as a random variable representing some part of the human face that takes values over the range of pixels and each column represents a sample from these mn random variables. This is a very complicated system and we want to compress and analyze the predominant patterns as best we can. To accomplish this we take the SVD of $\Phi = U\Sigma V^T$ where $U \in \mathbb{R}^{mn \times k}$. Now we have transformed these random pixel patterns of the column space into an ideal basis in the orthogonal matrix U. Instead of k human faces we now have k eigenfaces that represent the best basis for the human face! A quick visualization of some representative eigenfaces can be seen in Figure 2.

This is excellent but we can do much better than just finding the optimal basis, by examining the singular values in Σ we can evaluate how large of a contribution each eigenface makes to representing the human face. This is possible because in the matrix multiplication $U\Sigma V^T$ the matrix Σ acts to scale the columns of U and the rows of V^T by $\sigma_1, \sigma_2, ..., \sigma_k$. Therefore, if one of the singular values is large in comparison to all of the other ones then the associated eigenface in U must be more important. This idea can be formalized by examining

the fractional significance of the singular values:

$$f_j = \frac{\Sigma_{jj}}{\sum_{j=1}^{k} \Sigma_{jj}} \tag{4}$$

This simple equation simply takes each of the singular values in Σ and divides them by the sum of all the singular values. If we view the sum of all of the singular values as a scalar representation of the total amount of information in our system then f_i tells us the percent of the total information contained in a human face that eigenface u_i captures! This allows us to further reduce the ideal "face space" from k to some value $< k$ that accounts for enough of the pertinent information. For example we can say that if we can account for 95% of the variation in human faces then we only need to take the first $p < k$ eigenfaces where:

$$.95 \leq \frac{\Sigma_{jj}}{\sum_{j=1}^{p} \Sigma_{jj}}$$

If we pair this with Shannon entropy:

$$0 \leq d = \frac{1}{-\log(k)} \sum_{i=1}^{k} \Sigma_{ii} \log(\Sigma_{ii}) \leq 1 \tag{5}$$

we can acquire a concise estimate of how chaotic our data is. If d is close to 0 then our data has almost no entropy and the predominant patterns can be represented by a very small subset of our U matrix. This case could be represented by a Σ where $\sigma_1 = 1$ and $\sigma_2, ..., \sigma_k \approx 0$. However, if d is close to one then our data is very noisy and each eigenface has almost equal importance. This case correlates with a set of singular values where $\sigma_1 \approx \sigma_2 \approx ... \approx \sigma_k$. To this day there is still a lively debate over the minimal number of eigenfaces needed to completely represent face space but with the Shannon entropy we can shed some light on this issue and make a more informed decision on how many eigenfaces to include.

The size of our facial recognition and classification problem has been reduced from the number of pixels in an image to the number of images in our facial data set and then. If this

wasn't impressive enough, we then further reduced our problem by only taking the most important patterns by analyzing both the fractional significance and the Shannon entropy. This is a massive reduction in size! It is this reduction that makes recognition possible. We can begin to develop a system for actually classifying and recognizing faces. I will first demonstrate how facial classification can be achieved using our mathematical reduction and then proceed to demonstrate how this can be applied to recognizing a particular individual's face from our entire data set.

I like to use a geometric approach to explain how facial classification proceeds. Imagine that each eigenface represents a vector in $\mathbb{R}^{mn}$. Then together the set of all eigenfaces spans a $k-$cube in $\mathbb{R}^{mn}$, where k is the number of eigenfaces that we choose to represent our face space. The idea is that this k dimensional subspace encodes all of the pertinent information associated with a face, and furthermore it is the part of $\mathbb{R}^{mn}$ where any face will be located. Thus if we take any unknown image and look at how close it falls to this k-cube we can estimate how close it is to an ideal human face and how likely it is to actually be a face [**?**]. Now let's try and do this mathematically. The selected columns of U form our face space, and any new face vector Γ must fall somewhere in $\mathbb{R}^{mn}$. We want to know how close Γ is to our selected columns of U. We do this by projecting Γ into U and observing the

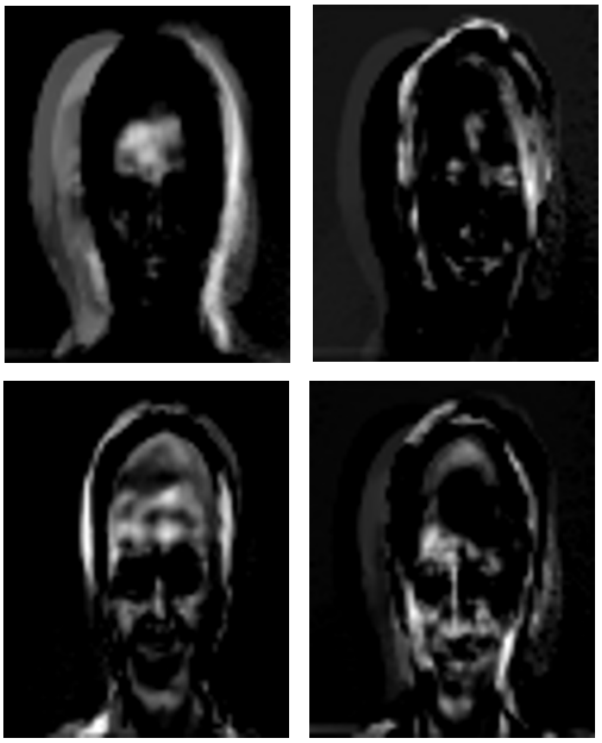

Figure 2: Four example eigenfaces [**?**]

outcome:

$$\omega = U^T \Gamma \tag{6}$$

The resultant vector ω represents the weights associated with each eigenface in forming the image Γ. In other words, if our image Γ was strictly a linear combination of the eigenfaces then:

$$\omega^T U = \omega_1 u_1 + \omega_2 u_2 + ... + \omega_k u_k = \Gamma' \tag{7}$$

Where, $\Gamma' = \Gamma + \epsilon$ represents the error in this approximation. This is necessary because naturally no image vector is going to fit perfectly into face space when every single pixel counts as a dimension in $\mathbb{R}^{mn}$. The idea is that if Γ is in fact a face then ϵ will be small because our face space should adequately account for all of the important characteristics. We now examine the error between our projection and the original image:

$$\|\Gamma - \Gamma'\| = \epsilon \tag{8}$$

If this epsilon is below some determined threshold α then we classify the image as a face. If it is not then we say that it is not a face. To illustrate this projection and reconstruction examine Figure 3. You can see clearly that the rose image when projected into face space looks nothing like the original. Interestingly it looks like a pixelated eigenface. This non facial image is represented poorly by eigenfaces but if we examine the picture of a human face the projection looks almost identical to the original image. This is because eigenfaces do a great job of representing human facial features. We have achieved a simple yet effective means of classifying an image of an unknown type as a face or not!

Now that we can classify an image as a face or not, let's try and do something a bit more exciting like recognize an unknown person's face using eigenfaces. Before we dive straight into

the mathematics let's try and develop a geometric intuition about what what will happen. We have this k-cube in $\mathbb{R}^{mn}$ where we believe human faces to lie. Now if we take some unknown face Ψ and project it into face space we get a line in our k-cube. If we project another known face into the face space we get another line. These two lines are some distance apart. If we have enough known faces and project all of them into our face space we then look for the projection that is closes to our unknown person's face. Now to formalize this notion.

Let's say that we have a picture of some unknown individual Ψ, and a database full of known faces Ω. We want to identify Ψ as best we can. We assume that our Ω contains a picture of the individual in Ψ and we want to see who they are. Well, this process is quite similar to recognizing an image as a face or not but with some slight variations. We begin by taking the unknown face and projecting it into the eigenface space as in equation (6). We now have ω_Ψ which represents the weights of each of the eigenfaces that contribute to the construction of Ψ'. We now calculate the projection of all of the faces in Ω into our eigenface space and get a set of weights $\omega_1, \omega_2, ..., \omega_n$. Next we calculate the euclidean error associate with ω_Ψ and all of the other projection weights. The one with the lowest error is the face in our database that is closest in face space to our unknown face. If that error is below some predetermined ϵ then we consider our facial recognition a success. If it is not then we classify the face as unknown.

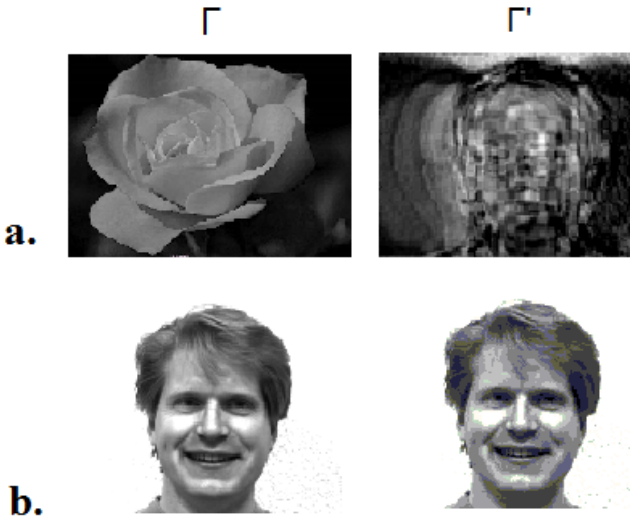

Figure 3: (a) A non-facial picture and its projection into face space. (b) A face and its projection into face space [**?**].

I will describe one more nice artifact of this method, facial compression. These days we have tons of data, and storing it can be a pain, especially large image databases. Imagine that we have a very very large set of face images that we want to store but not enough space to

store them. How can we keep the faces roughly the same but reduce the amount of memory that they occupy? Eigenfaces provide a solution! We have seen repeatedly how projecting a face into the eigenface space yields a vector ω which represents the weights of each eigenface. If we want to compress a large dataset of faces we can construct an eigenface basis using the SVD. Then we reduce the size of this basis by examining the Shannon entropy of the system and the fractional significance of the eigenvalues. Now that we have an eigenface space that is much smaller than the set of all images we can store any new face as a projection vector ω instead of an $m \times n$ pixel image. Let's think about this in terms of some actual numbers. Let's say we have 10,000 faces each of them is a 100 by 100 pixel image, this is very very small. That means that we have $100 \times 100 \times 10,000 = 100,000,000$ data points that we need to store! This is a lot of data. Now if we construct a face space, which usually will contain around 50 eigenfaces to characterize 95% of the data, then each image can now be stored as a weight vector which is only 50 data points. This makes the total size that we now have to store, $50 \times 10,000 + 50 * 10,000 = 1,000,000$ data points. That's two orders of magnitude fewer data points! To put it another way this new dataset is 1% the size of the original and contains 95% of the same information!

To summarize we have explored the power of a matrix decomposition and it's application to a very troubling problem. We find that the SVD's ability to reduce complicated patterns across both the columns and rows of the original matrix to an ideal basis is invaluable to data science and has wide applicability even among a single type of problem. This reduced space allows us to classify faces, recognize an unknown face, and even compress large quantities of facial data. All of this is possible with a remarkably simple method.

Below you can find the code that I used to generate most of my figures. The description of what is taking place is addressed directly in the comments:

```
  %Eigenface Generation
clear;
clc;

%Grab the directory with the faces we want to analyze
directory = 'C:\Users\tetracycline\Documents\MATLAB\Playthings\EigenfaceProject\Recogniti
files = dir(fullfile(directory,'*.jpg'));%place the faces into a structure array
faces = [];%initialize a
for k = 1:length(files)
    tempface = imread(strcat(directory,'\',files(k).name));%read in each face
    [~,~,di] = size(tempface);
    if(di ~= 3)
        grey = tempface;%grey scale it if it's not RGB
    else
        grey = rgb2gray(uint8(tempface));%grey scale it if it is RGB
    end
    %newface = grey(40:370,200:420);
    newface = grey;%have our grey face
    finalface = imresize(newface,[75,60]); %make the sizes consistent
    faceVector = finalface(:);%Vectorize the face
    faces = [faces faceVector]; %compile these vectors into a matrix
end
[handles.rows,handles.columns]=size(finalface);
handles.faces = faces;
handles.files = files;
handles.directory = directory;

%Generates orthonormal bases
```

```
[U,S,V] = svd(double(handles.faces));%Perform the SVD

%Extracts and reconstructs the eigenfaces
for  k = 1:handles.columns%Goes through each face column
    vector = U(:,k);%take the kth column
    j = 1;
    facefin = [];
    l = 1;
    while j < length(vector)
        index = handles.rows*l;
        face = vector(j:index);
        j=handles.rows * l + 1;
        facefin = [facefin face];
        l = l+1;
    end
    cellfaces(k) = {facefin};%create a cell array of the faces
end

% Normalization and eigenface generation
for k=1:length(files)

    adjustedIm = imadjust(cellfaces{k},stretchlim(cellfaces{k},[0.0001 0.9999]));%scale t
    resizedIm = imresize(adjustedIm,4);%resize the image
    eigenfaces(k) = {resizedIm};
    figure(k)%display them
    imshow(eigenfaces{k})

end
```

References

[1] N.M. Bertagnolli, R. Canning, J.M. Davis and D.J. Hoggatt, "Facial Recognition Using Singular Value Decomposition," *Proceedings of the University of Utah Conference on Machine Learning*, Dec. 2013.

[2] W.W. Bledsoe, "The model method in facial recognition," *Panoramic Research Inc.*, Palo Alto, CA, Rep. PRI:15, Aug 1966.

[3] G.H. Golub and C.F. Van Loan, *Matrix Computations.* Baltimore, MD: John's Hopkins University Press, 2013, pp.76-80.

[4] M. Kirby and L. Sirovich, Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 12, no. 1,pp.103-108, Jan. 1990.

[5] G.W. Stewart "On the early history of the Singular Value Decomposition," *SIAM Review*, vol. 35,no. 4, pp.551-566, Dec. 1993.

[6] M. A. Turk and A. P. Pentland, Eigenfaces for Recognition, *Journal of Cognitive Neuroscience*, vol. 3, no. 1,pp. 71-86, 1991

[7] Yale Face Database, [online] 2001, http://vision.ucsd.edu/content/yale-face-database (Accessed: 15 February 2013).