MTHE 493
Automated Facial Recognition

Kevin Gibson, Thomas Hughes, Mitchell Wasson

Supervisor: Professor Abdol-Reza Mansouri

April 2014

Acknowledgements

First and foremost we would like to acknowledge our supervisor, Professor Abdol-Reza Mansouri, whose support made this possible. We also would like to thank all the faculty, staff and students of the Math & Engineering program for the wonderful experience they have all helped to create.

Abstract

Several methods for facial recognition are presented and evaluated. First, Principal Component Analysis and Kernel Principal Component Analysis are reviewed. We then propose two novel techniques and test them on several different databases. A description of a method for generating a facial database from public Facebook photographs is given, and the effectiveness of all methods are tested on this new Facebook database.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Facial recognition is something that happens every time you interact with someone. Humans have an innate ability to recognize someone they have seen before. Attempts have been made since the 1960s to program computers to emulate this ability [1]. There are many applications that motivate the need to automate facial recognition.

The most prevalent industrial application of automated facial recognition today is in the field of security. Often firms concerned with access control to a physical area or system will implement a password system or have security personnel restrict access. Facial recognition (or other biometric) systems are often deployed in these situation to add an extra layer of security or to replace the existing techniques. Alternatively, consider the scenario where a security camera has captured a picture of a suspect at the scene of the crime. In order to find the suspect, a possible course of action is to distribute this picture to the public hoping someone would identify it; however, this trouble could be avoided entirely by having a computer compare the crime scene photos to a database of face images.

Recently, a diverse group of applications have arisen as well. A great social application is Facebook's attempts to integrate facial recognition into their photo tagging system to reduce the amount of human input needed. Alternatively, facial recognition technology could also be used to extract demographic information from video feeds of a public event.

Finally, facial recognition is of pure scientific interest as well. The ongoing quest to create true artificial intelligence, or intelligent machines, spans a wide range of different problems in computer science and mathematics. Facial and pattern recognition constitute an important subset of these problems. With this established need for automated facial recognition, we move forward to undertake the design of such a system.

1.2 Scope of Project

Functioning facial recognition systems are typically composed of multiple algorithms that perform specific tasks contributing to the recognition process. First, a set of training data is obtained. Before this data is used for recognition, it must be processed to provide optimal recognition conditions. An example of this preprocessing would be to crop and center the face images. After the training images have been suitably transformed, a modelling algorithm is applied in order to extract the necessary data. The extracted data is then stored in a database to be used for classification.

To classify an image, it is similarly subjected to the same preprocessing as the training images. Furthermore, the model used on the training images has a recognition component that is used here to extract the desired data from the test image. A classification algorithm will then use this newly acquired data along with the previously mentioned stored data to assign an identity to the image in question. This process is outlined in Figure 1.

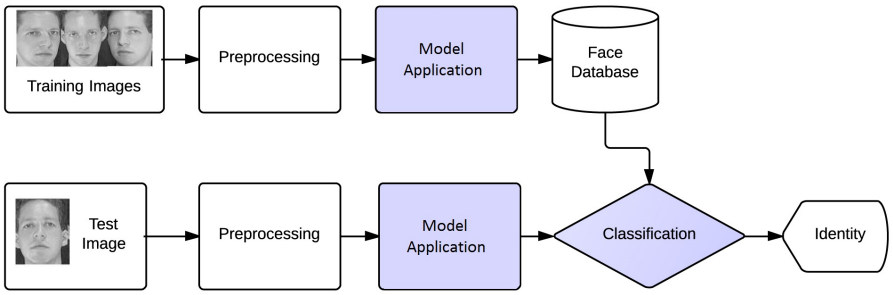

Figure 1: Abstract Facial Recognition Flowchart

In the confines of this project, we will assume that the necessary preprocessing has been performed on testing data. That is, we are only concerned with the modelling and classification stages of the facial recognition process. We aim to create modelling and classification algorithms that are robust to variations in training and testing data. In order to accomplish this task, we will first familiarize ourselves with the underlying math of many existing algorithms and rigorously analyze the problem. This will be followed by designing and implementing what we believe will be a robust algorithm. Finally, this algorithm will be compared with well know algorithms by testing on a variety of face databases.

2 Problem Analysis

2.1 Problem Description & Formalization

To approach the problem, we need a formal definition. Loosely put, the problem is to determine, given a set of images of individuals with known identities, if a new test image is an image of a face, and if this face corresponds to the identity of one of the training images. Mathematically, given a set of training faces $\{x_1, \ldots, x_T\}$ with known identities, determine the identity of a test face x.

An 8-bit grey-scale image with $m \times n$ pixels is an $m \times n$ matrix taking values in $\{0, 1, \ldots, 255\}$. For our purposes, we view the $m \times n$ matrix as a vector in $\mathbb{R}^{m \times n}$.

2.2 Principal Component Analysis

We are treating images as vectors in $\mathbb{R}^{m \times n}$. This space has a large dimension, and it is preferable to work in a lower dimensional space for computational reasons. In order to make facial recognition computationally feasible, it would be best to find a way to reduce the size of the spaces we are working with.

Using linear algebra and principal component analysis allows us to reduce the space by finding a subspace which approximately captures all the data points in the original space. First, we assume that an image is essentially a random vector of a high dimension. We will use principal component analysis to generate a set of orthonormal Eigenvectors from a set of training image vectors, and we can show that we can generate any of the training vectors from a linear combination of the Eigenvectors. Furthermore, principal component analysis shows that it is possible to reduce the dimension of the space by removing Eigenvectors while minimizing the loss of variance (information).

Let X denote a random vector with values in $\mathbb{R}^n$. More precisely, $X : \Omega \to \mathbb{R}^n$. Without loss of generality, we can assume that X is zero-mean. Let C be the covariance matrix of X, given by $C = E[XX^T]$. C is an $n \times n$ matrix, and from the properties of the covariance matrix, we have that C is symmetric and positive semi-definite.

Since C is a square $n \times n$ matrix, there are n corresponding Eigenvalues and Eigenvectors. Let $\{v_1, v_2, \ldots, v_n\}$ be the set of Eigenvectors of C with corresponding Eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. That is, $Cv_i = \lambda_i v_i \ \forall i \in \{1, 2, \ldots, n\}$. We also know this forms an orthonormal basis of $\mathbb{R}^n$. Hence, $\forall z \in \mathbb{R}^n, \exists! \{\alpha_i\}_{i=1}^n \subset \mathbb{R}$ such that $z = \sum_{i=1}^n \alpha_i v_i$. In particular, $\exists! \{\alpha_i\}_{i=1}^n$ real valued random variables such that

$$X(\omega) = \sum_{i=1}^{n} \alpha_i(\omega) v_i \ \ \forall \omega \in \Omega$$

i.e.

$$X = \sum_{i=1}^{n} \alpha_i v_i$$

Let $X_k = \sum_{i=1}^{k} \alpha_i v_i, \ \ k \leq n$. This is a reduction in dimension, which makes it easier to perform computations on. We would like to study how to minimize

the error between X and X_k. If we use the Euclidean norm,

$$\begin{aligned} E[\|X - X_k\|^2] &= E[\| \sum_{i=k+1}^{n} \alpha_i v_i \|^2] \\ &= E[\sum_{i=k+1}^{n} \|\alpha_i v_i\|^2] \\ &= E[\sum_{i=k+1}^{n} \alpha_i^2] \\ &= \sum_{i=k+1}^{n} E[\alpha_i^2] \end{aligned}$$

We see that the error is given by the coefficients in the linear expansion of all the dimensions that were not included in X_k. Furthermore, these coefficients are proportional to the Eigenvalues of the corresponding Eigenvectors.

$$\begin{aligned} X &= \sum_{i=1}^{n} \alpha_i v_i \\ \Rightarrow v_i^T X = \alpha_i &= X^T v_i \\ C v_i &= \lambda_i v_i \\ C = E[XX^T] & \\ \Rightarrow v_i^T C v_i &= v_i^T E[XX^T] v_i \\ &= E[(v_i^T X)(X^T v_i)] \\ &= E[\alpha_i^2] \\ \Rightarrow E[\alpha_i^2] &= \lambda_i \end{aligned}$$

So we see that

$$E[\|X - X_k\|^2] = \sum_{i=k+1}^{n} \lambda_i$$

So, we have a method for reducing the dimensionality of a set of data, and minimizing the error. By choosing the k Eigenvectors of C with the largest corresponding Eigenvalues, we can create the best possible partial basis for the given data. These Eigenvectors are known as the principal components.

2.3 Eigenface

So the question remains: how does principal component analysis relate to facial recognition? In their 1991 paper, Turk and Pentland outlined their process for automated facial recognition, which used the principal components of a set of training images to classify and later identify new faces introduced to the system

[2]. By projecting an image onto the face spanned by the principal components of a known set of face images, they argued, it would be possible to represent an identity of a person by a "weight pattern" representing the linear projection onto the principal components.

Let $x_1, x_2, \ldots, x_T$ be a set of $n \times m$ training images. That is, $x_i \in \mathbb{R}^N := \mathbb{R}^{n \times m} \ \forall i \in \{1, 2, \ldots, T\}$. Since data must have a mean of zero in order for principal component analysis to work, an average vector is computed. Let $\Psi = \frac{1}{T}\sum_{i=1}^{T} x_i$ be the average face vector, and $\Phi_i = x_i - \Psi$ be the difference faces, $\forall i = \{1, 2, \ldots, T\}$. We wish to use principal component analysis to find a representation of these N dimensional vectors in a much smaller dimension, and it turns out we can, quite easily. Our derivation of PCA uses a covariance matrix. Since we do not have the distribution of the faces, we use a sample covariance matrix,

$$C = \frac{1}{T}\sum_{i=1}^{T} \Phi_i \Phi_i^T = AA^T$$

where $A = [\Phi_1 \Phi_2 \ldots \Phi_T]$. However, finding the Eigenvectors of this $N \times N$ matrix would be computationally expensive for most typical image sizes, so another method must be found. Fortunately, we can use elementary linear algebra to reduce our calculations to finding the Eigenvectors of a $T \times T$ matrix - much smaller problem.

Let v_i be the Eigenvectors of $A^T A$, with A as defined above, and $\{\mu_1, \ldots, \mu_T\}$ their Eigenvalues. Then

$$\begin{aligned} (A^T A)v_i &= \mu_i v_i \\ A(A^T A)v_i &= A(\mu_i v_i) \\ AA^T(Av_i) &= \mu_i(Av_i) \end{aligned}$$

So we get that Av_i are the Eigenvectors of the matrix AA^T, and we have reduced the problem of finding Eigenvectors of AA^T to that of finding the Eigenvectors of $A^T A$. Moreover, we can order the Eigenvectors in terms of their corresponding Eigenvalues, and determine which vectors are most critical in capturing the features of the training faces.

Now that we have found the Eigenvectors of $A^T A$, Eigenfaces are generated by

$$u_l = \sum_{k=1}^{T} v_k \Phi_k, \qquad l = 1, 2, \ldots, T$$

These Eigenfaces essentially form a basis with which we can represent new face images. As shown in principal component analysis, it is possible to choose the $M \leq T$ Eigenfaces with the greatest Eigenvectors, and minimize the loss.

So, if given a new face image x, it can be classified as a series of scalar weights computed by the projection of x onto each Eigenface:

$$\omega_k = u_k^T(x - \Psi), \qquad \forall k = 1, \ldots, M$$

In a slight semantic abuse, these projections are known as the principal components of the test vector x. The problem of facial recognition has been reduced to the problem of comparing the distance between a vector $\omega = [\omega_1, \omega_2, \ldots, \omega_M]$, representing the unknown face as a vector of its principal component vector, with the principal components of the training faces, $\{\omega^k\}_{k=1}^T$, where $\omega^k = [\omega_1^k, \omega_2^k, \ldots, \omega_M^k]$. There are a variety of tools available for this comparison. The simplest is the Euclidean distance between ω and and the various ω^k. We discuss classification methods further in Section 3.2.

2.4 Kernel Principal Component Analysis

PCA and its use in Eigenface has been shown to be an effective method in facial recognition [2]. Its primary working assumption is that the space around which facial images cluster, the "face-space," is linear. This is implicit in the subspace being modelled as the linear span of the most significant Eigenvectors. Linearity is both a good and limiting assumption. It greatly simplifies the model, and makes computations simpler. For example, computing the projections of the training and test faces onto the face-space was possible due to the linearity of the space. Generally, nearest neighbor rules are difficult to evaluate on non-linear manifolds.

On the other hand, linearity is a somewhat inaccurate assumption. Linear models make good approximations, but it is unlikely that a space as complex as the space of facial images is strictly linear. It is desirable to choose methods that model the face-space non-linearly. Methods do exist for explicitly modelling the non-linearity, known as principal curve methods [6]. These can be executed effectively via neural networks, but attention must be given not to overfit the general trend. However, instead of explicitly modelling the space, we opt for a different approach.

Suppose that the space of faces is $\Lambda \subset \mathbb{R}^N$. The idea is to define a mapping $\psi : \mathbb{R}^N \to \mathbb{R}^L$, with $L \gg N$ such that $\psi(\Lambda)$ is linear in $\mathbb{R}^L$. We will call $\mathbb{R}^L$ the feature space, and note that L is potentially infinity. In the feature space, we perform PCA on the transformed vectors, for which the same theoretical work from Section 2.2 applies. We present a derivation of this new method similar in structure to Scholkopf et al[5].

Suppose we have test faces $x_1, \ldots, x_T$, and a transformation $\psi : \mathbb{R}^N \to \mathbb{R}^L$ under which the face space is transformed to a linear space. The transformed face vectors are $\psi(x_1), \ldots, \psi(x_T)$. We assume that the transformed vectors are zero mean in the feature space, i.e. $\frac{1}{T}\sum_{i=1}^T \psi(x_i) = \mathbf{0}$. This is an untrue

assumption, but we can modify the algorithm to center the vectors in the feature space. We include this in Appendix A. Now, consider the empirical covariance matrix of the data

$$\overline{C} = \frac{1}{T}\sum_{i=1}^{T} \psi(x_i)\psi(x_i)^T$$

As in the Eigenfaces method, we wish to use the Eigenvectors of this matrix, which will satisfy

$$\overline{C}V = \lambda V$$

It is very important that for each such Eigenvector V, $V \in \text{Span}\{\psi(x_1), \ldots, \psi(x_T)\}$. So we can write, for some constants $\alpha_1, \ldots, \alpha_T$,

$$V = \sum_{i=1}^{T} \alpha_i \psi(x_i) \tag{1}$$

We can also, by simple substitution of the Eigenvalue equation (2.4), write

$$\lambda(\psi(x_k) \cdot V) = (\psi(x_k) \cdot \overline{C}V) \tag{2}$$

Combining (1) and (2) gives the following:

$$\begin{aligned}
\lambda(\psi(x_k) \cdot V) &= \lambda(\psi(x_k) \cdot \sum_{i=1}^{T} \alpha_i \psi(x_i)) \\
&= \lambda \sum_{i=1}^{T} \alpha_i (\psi(x_k) \cdot \psi(x_i)) \\
&= \sum_{i=1}^{T} \alpha_i (\psi(x_k) \cdot \overline{C}\psi(x_i)) \\
&= \sum_{i=1}^{T} \alpha_i \left[\psi(x_k) \cdot \left(\frac{1}{T} \sum_{j=1}^{T} \psi(x_j)\psi(x_j)^T \right) \psi(x_i) \right] \\
&= \frac{1}{T} \sum_{i=1}^{T} \alpha_i (\psi(x_k) \cdot \left(\sum_{j=1}^{T} \psi(x_j)(\psi(x_j) \cdot \psi(x_i)) \right) \qquad (3)
\end{aligned}$$

$$\text{for all } k = 1, \ldots, T.$$

If we define a $T \times T$ matrix K of feature space dot products,

$$K_{ij} := (\psi(x_i) \cdot \psi(x_j))$$

then, letting $\alpha = [\alpha_1, \ldots, \alpha_T]^T$, we can rewrite (3) as

$$T\lambda K\alpha = K^2\alpha \tag{4}$$

Symmetry of K implies it has a set of Eigenvectors spanning the space, so

$$T\lambda\alpha = K\alpha$$

gives us the complete set of solutions to (4). Furthermore, K is positive semi-definite. We also wish to have our feature space Eigenvectors $\{V^k\}_{k=1}^T$ normalized. The condition $(V^k \cdot V^k) = 1$ is achieved by requiring that $1 = \lambda_k(\alpha^k \cdot \alpha^k)$. For these derivations, we refer to [5].

Now, to use this method for classification, we need to compute the principal components of a test vector in the feature space, i.e. the projections of a test vector onto the feature space Eigenvectors V^k. The k^{th} non-linear principal component ω_i of a test vector x may be computed in the feature space as follows

$$\omega_k = (\psi(x) \cdot V^k) = \sum_{i=1}^{T} \alpha_i^k (\psi(x_i) \cdot \psi(x)) \tag{5}$$

Suppose we keep the M most significant Eigenvectors in the feature space. To classify x, we compute its principal components $\omega = [\omega_1, \ldots, \omega_M]$. The vector ω can then be compared with the principal components of the training faces, and associated with the nearest one.

Until now, we have largely left some major points unaddressed; we have not specified our transformation ψ, nor the dimension L of the feature space. Furthermore, without these data, we have no means of computing feature space dot products of the form $(\psi(x) \cdot \psi(x))$. In much the same way that it is difficult to explicitly model the non-linear face manifold, it is also difficult to specify a transformation that will linearize this manifold. This technique of non-linear PCA is called Kernel Principal Component Analysis because to avoid these problems, it defines ψ implicitly via kernel functions. One might wonder how, then, without knowing our transformation, we will know the representations of our vectors in the feature space. The answer is that we will not; by choosing our kernels to satisfy Mercer's theorem, from functional analysis, we can compute dot products in the feature space via the kernel function, without knowing the the transformed vectors.

Theorem 2.1 (Mercer's Theorem)**.** *If k is the continuous kernel of a positive definite integral operator $\mathcal{K}$, then it defines a transformation ψ for which*

$$(\psi(x) \cdot \psi(y)) = k(x, y) \tag{6}$$

Proof. Let $k(\cdot, \cdot)$ be a continuous kernel of integral operator $\mathcal{K} : \mathcal{L}_2 \to \mathcal{L}_2$,

$$(\mathcal{K}f)(y) = \int f(x)k(x, y)dx$$

such that $\mathcal{K}$ is positive definite, i.e.

$$\int\int f(y)k(x,y)f(x)dxdy > 0, \text{ if} f \neq 0.$$

Then we can expand k into a series

$$k(x,y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(y)$$

where $\lambda_i \geq 0$ and $(\phi_i \cdot \phi_j)_{\mathcal{L}_2} = \delta_i^j$. Defining a transformation

$$\psi(x) = \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(x),$$

it holds that

$$(\psi(x) \cdot \psi(y)) = k(x,y)$$

□

So, if we choose to define ψ via kernel functions, and we do so with kernel functions satisfying Mercer's Theorem, then using relation (6) we can compute the matrix K as

$$K_{ij} = k(x_i, x_j) \tag{7}$$

and our principal component equation (5) becomes

$$\omega_k = (\psi(x) \cdot V^k) = \sum_{i=1}^{T} \alpha_i^k k(x_i, x) \tag{8}$$

This is the reason for the earlier derivation going to great lengths to express everything solely in terms of feature space dot products. Kernels that have been shown to satisfy Mercer's Theorem and that are commonly used are Gaussians, polynomials, and sigmoids [5][6]. Respectively,

$$k(x,y) = e^{-\|x-y\|^2/\sigma^2}$$
$$k(x,y) = (x \cdot y)^d$$
$$k(x,y) = \tanh(a(x \cdot y) + b)$$

The choice of kernel as well as determining optimal parameters will be discussed in Section 3.

2.5 Principal Class Manifolds

In the previously mentioned approaches to facial recognition, principal manifolds are used to reduce the dimension of computation by attempting to discard useless information. In these approaches, the principal manifold is constructed for the entire face space. Once this manifold has been obtained, movements

inside the manifold are interpreted as interperson variations and movements outside the manifold as intraperson variations (e.g., lighting, emotion).

Instead, in this section we propose modelling each person in the training database with their own principal manifold. By doing so, movement along inside the manifold of a given person is interpreted as intrapersonal variation for the given subject. Similarly, moving outside the manifold of a given person, or between different subjects' manifolds, corresponds to interpersonal variation.

2.5.1 Linear Principal Class Manifolds

Here we outline the use of PCA to construct principal hyperplanes that model each person individually. Let $\{x_{ij} : i \in \{1, 2, \ldots, P\}, j \in \{1, 2, \ldots, T\}\}$ be the set of training images where T is the number of training images per person and P is the number of people in the training set. That is x_{ij} is the j^{th} training image of the i^{th} person.

We start by computing the average vector for each person $\Psi_i = \frac{1}{T}\sum_{j=1}^{T} x_{ij}$. We can then calculate the class specific difference faces $\Phi_{ij} = x_{ij} - \Psi_i$ where $i \in \{1, 2, \ldots, P\}$ and $j \in \{1, 2, \ldots, T\}$. The set of difference faces for each person are now zero mean. The process outlined in Section 2.3 is then applied to each person individually to find the feature vectors of each person. These are the orthonormal Eigenvectors of the empirical covariance matrices

$$C_i = \frac{1}{T}\sum_{j=1}^{T} \Phi_{ij}\Phi_{ij}^T \qquad i \in \{1, 2, \ldots, P\}$$

Let $\{u_{ij} : i \in \{1, 2, \ldots, P\}, j \in \{1, 2, \ldots, T\}\}$ be the set of feature vectors mentioned above, and for person i, let $\{u_{i1}, u_{i2}, \ldots, u_{iT}\}$ be sorted in order of decreasing Eigenvalue. We now only keep the first $M \leq T$ Eigenvectors for each person in order to model each person with an M-dimensional hyperplane. The essence of this process is outlined in Figure 2 where a line of best fit is used to describe each class.

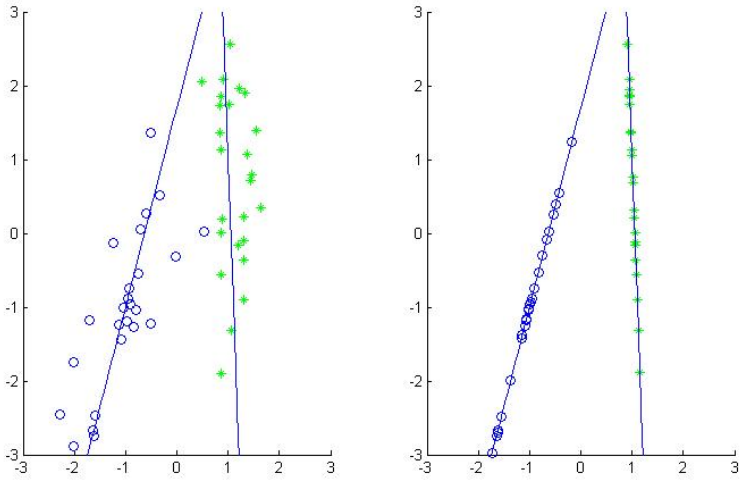

Figure 2: Linear Principal Class Manifold Illustration

At this point the training images can be discarded. We only need to keep the set of average vectors $\{\Psi_i : i \in \{1, 2, \ldots, P\}\}$, as well as the set of feature vectors $\{u_{ij} : i \in \{1, 2, \ldots, P\}, j \in \{1, 2, \ldots, M\}\}$.

Given a new face image x, the simplest way to classify it is to identify with the closest plane. If the u_{ij} are unit, we calculate the distance of x to the different principal hyperplanes as

$$d_i = (x - \Psi_i)^T (x - \Psi_i) - \sum_{j=1}^{M} (x - \Psi_i)^T u_{ij} \qquad i \in \{1, 2, \ldots, P\}$$

x is then identified with the identity of $\operatorname{argmin} d_i$.

In the Eigenface method, the feature vectors describe the entire space of faces. When visualized as images, these vectors roughly look like faces. The feature vectors obtained in Linear Principal Class Manifolds describe the space of one person's face. Therefore, when visualized as images, these vectors look like the face of the person in question. Figure 3 displays this difference by showing the four most significant feature vectors of the AT&T face database from the Eigenface method, as well as the four most significant feature vectors from the principal hyperplane of a single person.

Figure 3: Feature Vector Comparsion (Eigenface Vectors on Left)

2.5.2 Kernel Principal Class Manifolds

The underlying premise of Kernel Principal Class Manifolds is that even each person will not cluster around a linear manifold. Here we will outline the use of KPCA to construct principal hyperplanes in transform spaces that model each person individually. As described in Section 2.4 one can implicitly define a non-linear mapping that will "flatten out" a set of data by the use of kernel functions.

Again, let $\{x_{ij} : i \in \{1, 2, \ldots, P\}, j \in \{1, 2, \ldots, T\}\}$ be the set of training images where T is the number of training images per person and P is the number of people in the training set. It is unreasonable to assume that the same mapping will "flatten out" the set of images for each person equally well. Therefore, each person will have their own mapping $\psi_i : \mathbb{R}^N \to \mathbb{R}^{L_i}$ implicitly defined by a kernel function satisfying $k_i(x, y) = (\psi_i(x) \cdot \psi_i(y))$.

We perform KPCA on the training images $\{x_{ij} : j \in \{1, 2, \ldots, T\}\}$ with the kernel function k_i, $i \in \{1, 2, \ldots, P\}$. By doing so we obtain the vectors $\left\{\alpha^{i1}, \alpha^{i2}, \ldots, \alpha^{iT}\right\}$ where the j^{th} feature vector of person i, V^{ij}, can be expressed as

$$V^{ij} = \sum_{k=1}^{T} \alpha_k^{ij} \psi_i(x_{ik})$$

Recall, these feature vectors are the orthonormal Eigenvectors of the covariance matrices

$$\overline{C}_i = \frac{1}{T} \sum_{j=1}^{T} \psi_i(x_{ij}) \psi_i(x_{ij})^T$$

Similarly, we only keep $M \leq T$ of these feature vectors to model each person with an M-dimensional hyperplane in their respective feature spaces. Equivalently, we keep $\left\{\alpha^{i1}, \alpha^{i2}, \ldots, \alpha^{iM}\right\}$ for each person, since these feature vectors are never explicitly computed.

Now, given a new face image x, again the simplest way to classify this image is to identify with the closest manifold. We do not have the parametrizations for the manifolds defined by the kernel mappings $\{k_1, k_2, \ldots, k_P\}$. Because of this we will define the distance of x to each person as the distance of x to said person's principal hyperplane in their feature space. In this section we are assuming that the training data is centered by the mappings. With this assumption, and the orthonormality of the feature Eigenvectors, these distances are calculated as

$$\begin{aligned} d_i &= (\psi_i(x) \cdot \psi_i(x)) - \sum_{j=1}^{M} (\psi_i(x) \cdot V^{ij}) \\ &= k(x,x) - \sum_{j=1}^{M} (\psi_i(x) \cdot \sum_{k=1}^{T} \alpha_k^{ij} \psi_i(x_{ik})) \\ &= k(x,x) - \sum_{j=1}^{M} \sum_{k=1}^{T} \alpha_k^{ij} (\psi_i(x) \cdot \psi_i(x_{ik})) \\ &= k(x,x) - \sum_{j=1}^{M} \sum_{k=1}^{T} \alpha_k^{ij} k(x, x_{ik}) \end{aligned}$$

Generally, the kernel does not center the training data in the feature space of a person. Because of this, we must incorporate centering into our KPCA calculations as mentioned previously. We must also center the new face x when computing distances. These procedures are outlined in Appendix A.

Once these distances are calculated, x is identified with the person whose principal manifold is nearest to $\psi(x)$.

3 Design

Choosing a mathematical model for the system and deriving algorithms that exploit it is a major part of the facial recognition problem. It requires a lot of insight and mathematical knowledge. However, the problem is not solved at this point. Once the framework is laid out, there are numerous other design considerations, parameters to optimize and processes to tweak before the system is complete, and to make it robust to dataset variation. Here we discuss these.

3.1 Optimization

Many of the design considerations relate to choosing kernels and optimizing their parameters for KPCA. Additionally, we wanted to choose optimal subspace dimensions for our Eigenspaces for all the methods.

The problem of choosing an Eigenspace dimension is fairly simple. We wish to keep enough basis vectors such that we achieve peak recognition, but we do not want any extra, since this adds needless memory and computation (and sometimes degrades performance). Good dimensions were more or less chosen by trial and error. We performed recognition for a wide range of dimensions, and observed where the marginal increase in performance per added face essentially vanished.

Using KPCA for recognition introduces a range of new design aspects not present in normal PCA. Specifically, these aspects are the choice of kernel functions and their parameters. At the end of Section 2.4, we mentioned that commonly used kernels satisfying Mercer's theorem are Gaussians, polynomials and sigmoids. Furthermore, it is true that linear combinations of these functions also satisfy Mercer's theorem. In [7], experimental results were best for the Gaussian kernel. They used a generalized version of the Gaussian kernel that substituted a diagonal covariance matrix for the scalar variance, allowing different variances for each component. However, this path of investigation seemed quite well-trodden. Their comparison of these kernels, as well as their linear combinations, was thorough. Since we were more interested in the performance of the principal class manifold methods relative to the traditional ones, we chose to focus on the Gaussian kernel for KPCA.

Still, we had to optimize the variance parameter. In the interest of not repeating the work in [7], we used a normal scalar variance and not their generalization. The transformation specified by the kernel is unknown, and thus estimating the variance value that will best linearize the data set cannot be done analytically. Rather, we heuristically optimized the variance. MATLAB has a built-in function *fminsearch* which searches for parameters minimizing a given quantity. Iterating to maximize recognition rates would have required many repeated iterations of performing our recognition algorithm on the entire data set, a lengthy procedure. Instead, recall that PCA methods allow for comparison of basis vectors via their Eigenvalues, and that we discard Eigenvectors with lower Eigenvalues. In this spirit, we chose to search for variances minimizing the ratio between the discarded Eigenvalues and the kept Eigenvalues. The intention here was to maximize the amount of information we kept for a given dimension size. This optimization was therefore Eigenspace dimension-specific, but past a certain point the optimal variance was the same for different dimensions.

The framework we laid out for Kernel Principal Class Manifold recognition uses the theory for KPCA, including the use of kernel functions satisfying

Mercer's Theorem. However, since this method relies on distances to different feature subspaces instead of the comparison of non-linear principal components with those of training faces, it isn't necessarily true that the same kernels will be effective. Our results confirmed this. We found that using class manifolds with Gaussian kernels did not work. Recognition rates were on the same order as randomly guessing. However, we found that odd degree polynomials worked fairly well. The best polynomial was degree 1, which reduces to standard PCA. This suggested that non-linear class manifolds may not improve on linear class manifolds. To further investigate, we proposed a new kernel function of the form

$$k(x, y) = e^{(x \cdot y)/\sigma^2}$$

We will call this the exponential kernel. Note that the interaction of x and y is via a dot product and not a distance squared, as in the Gaussian. If we consider the series expansion of the exponential function, this is in some sense an infinite linear combination of polynomial kernels. We emphasize that that this kernel has not been shown to satisfy Mercer's Theorem. However, exponential kernel recognition rates with non-linear class manifolds were as good as or better than linear class manifolds. The use of the exponential kernel, its viability, and questions it raises, will be further discussed later on.

3.2 Classification

Much of the discussion of our methods in Section 2 focused on creating an appropriate Eigenbasis of our space and computing the principal components $\omega = [\omega_1, \ldots, \omega_M]$ of a test face. Prior to classification, it is understood that we have computed the principal components of each of our training images. For the i$^{\text{th}}$ training face, we denote this $\omega^i = [\omega_1^i, \ldots, \omega_M^i]$. The problem of classification is concerned with choosing which of $\omega^1, \ldots, \omega^T$ our test principal component vector ω is *closest* to.

The most obvious choice of such a distance measure is the Euclidean norm, i.e. if $\arg\min_i \|\omega - \omega^i\| = j$, we identify our test face with the identity of the j^{th} training face. This is known as a nearest neighbour classification rule, with Euclidean distance. A generalization of this is the k^{th} nearest neighbour classification rule. For an odd integer k, we find the k nearest training faces, in terms of principal components, and identify the test subject with the person to whom belong the most of the k^{th} nearest neighbours. Suppose that $L_{k\text{NN}}$ is the probability of error, i.e. misclassification, of a k^{th} nearest neighbour classification rule, as $T \to \infty$, where T is the number of data points (in our case training faces). Then it is true that

$$\cdots \leq L_{(2k+1)\text{NN}} \leq L_{(2k-1)\text{NN}} \leq \cdots \leq L_{3\text{NN}} \leq L_{1\text{NN}}$$

[8] This indicates that in theory, if we have arbitrarily many training faces, the more nearest neighbours we use for classification, the better our results. We

attempted this, trying various values of k for our simulations. However, the asymptotic nature of the above result does not appear to hold for finite data cases. We found that increasing the number of nearest neighbours used in classification strictly reduced our recognition rates. We attribute this discrepancy with the theory as being due to our finite and fixed T, in contrast with the theorem requiring that $T \to \infty$.

Beyond simple nearest neighbour methods, there are more sophisticated techniques for classification. One such method is the Mahalanobis distance. This method models the distribution of each subject as a multivariate Gaussian, and accordingly measures distance probabalistically [8]. However, this too performed worse than nearest neighbour classification in our simulations. In the end, we proceeded to use a simple nearest neighbour classifier, due both to its simplicity and apparent superiority.

3.3 Facebook

While researching various facial recognition systems, it became clear that having a proper testing database was necessary to evaluate the effectiveness of our designs. In order to allow us to observe trends across varying dimensionality, a large number of photos was required. As well, in order to test the robustness of our algorithm to high-variation data, wide variety in the photographs was desired. After investigating existing options, the lack of satisfactory databases led us to examine the possibility of generating our own, using photos available on Facebook. With a combination of Facebook's application programming interface (API) and open source facial detection software, we were able to automatically generate a set of facial photos of a wide variety of subjects.

Facebook developed the Graph API to developers to allow third-party applications to access user data and to allow features such as posting status updates from non-Facebook websites. It exposes the incredible amount of data Facebook has as an undirected graph, with nodes representing different Facebook objects, such as comments, users, or photos. Photo nodes contain a link to the original image file, as well as the names and locations of any people who were 'tagged' by users in the photo. With some code written in C#, it was possible to traverse a list of users, get a list of all photos they had been tagged in, download it along with the tag information. While the list of users was chosen to be one of our complete list of "friends", the photos obtained were limited to those that are publicly available.

Once the photos had been downloaded, processing was required to isolate the faces of subjects. This was done using a combination of the face tag data that was obtained from Facebook, and an open source facial detection algorithm from the OpenCV package. OpenCV has a set of built-in algorithms, including a Haar classifier method, designed to detect faces. While the details of this algorithm are outside the scope of this project, the code essentially re-

turns an axis-aligned bounding rectangle around any faces detected in a given photograph. While the accuracy of this algorithm is not ideal, it proved to be satisfactory when combined with the existing user-defined Facebook tag data. Facebook tags are given as a single point on the plane of the image. To identify the faces recognized by the Haar classifier, we simply match the rectangle given to the facebook tag point which lies inside it. If more than one point tag lies inside the rectangle, we simply discard that face - the large amounts of data we had meant that perfect facial extraction was not necessary.

Letting the program run for an extended period of time, we were able to obtain a database of around 60,000 photos, each labelled with an anonymous identifier. The size of the database allowed the running of much larger tests, making any differences in accuracy more statistically significant, as a change in accuracy of a few percent meant the recognition of several hundred faces. The method of collection meant that the number of photographs obtained for each subject was uncontrolled. This meant that certain subjects had many more photographs available than others, so for testing we limited ourselves to a fixed number of photographs per subject, and simply chose a random subset of photos from the ones available to us. Figure 4 shows the distribution of the number of photographs available per subject. Note that the number of subjects with less than 25 photos was left off for clarity.

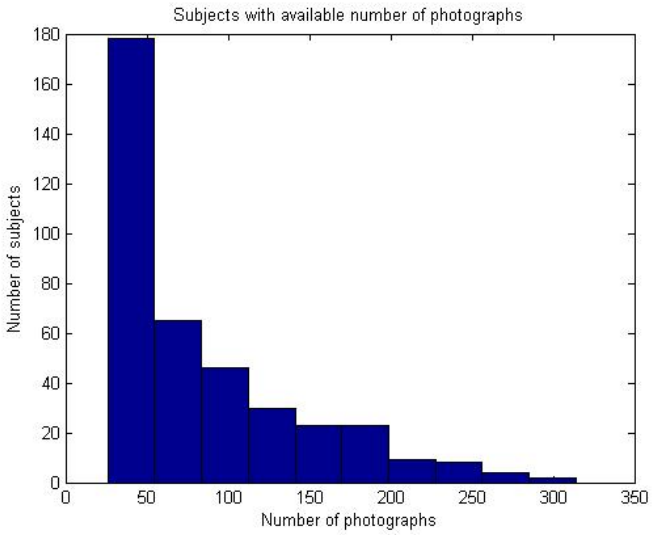

Figure 4: The distribution of the number of photos available per subject in the Facebook database

As well, the unsupervised nature of the photo collection means that the quality of the photographs for facial recognition is considerably reduced compared

to the more carefully constructed databases like the AT&T or Yale databases. For example, the Haar classifier frequently has false positives, identifying areas as faces when they do not contain them. Combining this with the untrustworthy nature of user generated Facebook tags, the result is that many images obtained are either from undesirable angles, are partially obstructed, or have less than ideal lighting conditions (see Figure 5, below). Despite these limitations, the size of the database allowed for constructive testing, bearing in mind the low recognition rates resulting from the variances in the data. It is useful to study facial recognitions on such "unreliable" databases, as more and more large sources of information are uncontrolled social networks.

3.4 Engineering Tools

In any engineering design problem, it is important to use the appropriate tools. Our problem was algorithmic and therefore the tools we have to consider are our choices of software, programming languages, and libraries.

We chose to use MATLAB for the primary implementation of all the methods. This was for several reasons. First of all, writing programs in MATLAB is much easier than lower level languages, so it is excellent for prototyping. This simplicity helped us the whole way through. Next, the majority of the computation involved in PCA and KPCA is matrix multiplication and dot product evaluation. This is what MATLAB is built for; it is completely optimized for matrix operations. Finally, our project obviously involves a fair amount of data I/O, specifically importing images. MATLAB has very simple and fast image importing functions that made for a much easier time than lower level languages would have.

For the construction of the Facebook database, the C# programming language was used, due to previous experience with making web requests using the language. As the amount of code needed was relatively small, and only needed to be run once, the primary requirement was to be able to implement it quickly. C#'s high level features, and Microsoft's .NET library meant that getting a working system required minimal code. Unfortunately, using C# and the .NET library limited us to using Windows machines; however, since the code only needed to be run once to generate the database, this was not a problem. The Facebook Graph API was accessed using HTTP requests, and the returned data was parsed in the C# program. To process the photos, the OpenCV library was used, which was chosen due to its large community, making finding documentation on using the facial detection algorithm simple.

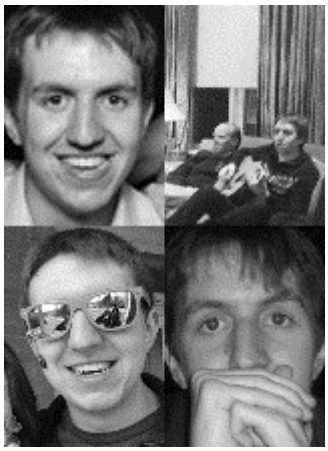

Figure 5: A sample of photos of one subject from the Facebook database.

4 Testing

4.1 Methodology

After a thorough analysis of the problem and rigorous design procedure, the next step was to test our algorithms. Coarsely, facial recognition systems are successful on the basis that they can achieve high recognition rates. However, there are several specific veins of investigation that were of interest to us, which we chose to investigate alongside recognition rates. An obvious one is how the techniques' performance varies with Eigenspace dimension, i.e. the number of Eigenvectors we keep, and project onto. We expect all the techniques' recognition rates to plateau at some point - it is interesting to compare these points, if they differ. If one requires a larger Eigenspace to reach peak performance, then this algorithm has somewhat larger storage requirements and larger computations, important factors when choosing an algorithm. Studying the storage requirements is a closely related problem, but also involves the storage of mean vectors. A feature of the principal class manifold methods is that they require we keep at least one Eigenvector per person, because every subject needs a space of at least dimension one. They also require a mean vector for every person. This is why the relationships of dimension and storage with recognition rates were of particular interest to us.

Another parameter we tested was the sensitivity of our techniques to the number of available training faces. Generally, we expect all techniques to improve the more training faces there are, as our Eigenspace should then be a better representation of the face space. This is also interesting with specific regard to our principal class manifold methods, because they require at least two training faces *per subject* to build a personal Eigenspace (it is two and not

one because we subtract the mean, and subtracting the mean of one vector gives zero).

Finally, we wanted to test the robustness of our algorithms to non-uniformity in the database, and clustering of the face space. Non-uniformity refers to a high variance database, with *imperfect* photographs: this includes changes in lighting, facial expression, angle, background, etc. Different facial recognition databases are known for their different amounts of variation. Clustering of the face space refers to a large number of different subjects. For a fixed number of pixels in our images, the more subjects we have, there will be more subjects whose principal components are near to someone else's. This will obviously reduce recognition rates across all methods. However, it was our hope that the principal class manifold methods would be more robust to high variation databases, and clustering. Testing this was a major motivation for us to build the Facebook database; the AT&T and Yale databases were lower in variation than we were interested, as well as containing too few subjects for real clustering to occur.

4.2 AT&T Faces

The first set of data that we tested our algorithm on is the AT&T Face Database originally outlined in [3]. This database is composed of forty subjects with ten 92×112 greyscale images each. We can see in Figure 6 that these images are well processed and have little variation in lighting, emotion, and other parameters. The purpose of testing with such a database is see if our algorithm performs in scenarios where traditional algorithms are successful.

Figure 6: Sample of AT&T Database Images

The first test we performed was to observe the effect of Eigenspace dimension on recognition. For traditional PCA methods, the variable is simply the dimension of the entire Eigenspace, or the number of kept Eigenvectors. For Principal Class methods, the variable is the dimension of each person's personal Eigenspace, the number of kept Eigenvectors *per person*, which explains the high recognition rates for low dimensions. We performed this test for all different methods, and here we include the results for Eigenface and Linear Principal Class Manifolds in Figure 7, and we note that the respective kernel implementations showed similar trends.

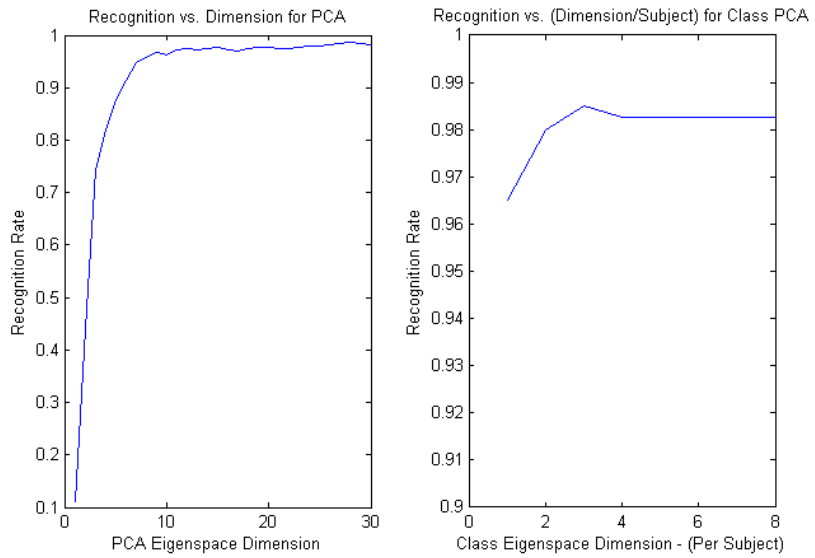

Figure 7: Eigenspace Dimension and Recognition

The next test performed on this database was to observe the effects of the number of training images of recognition rates. This test was performed on all algorithms, and the results of this test for Eigenface and Linear Principal Class Manifold implementations with nearest neighbour classification are show in Figure 8.

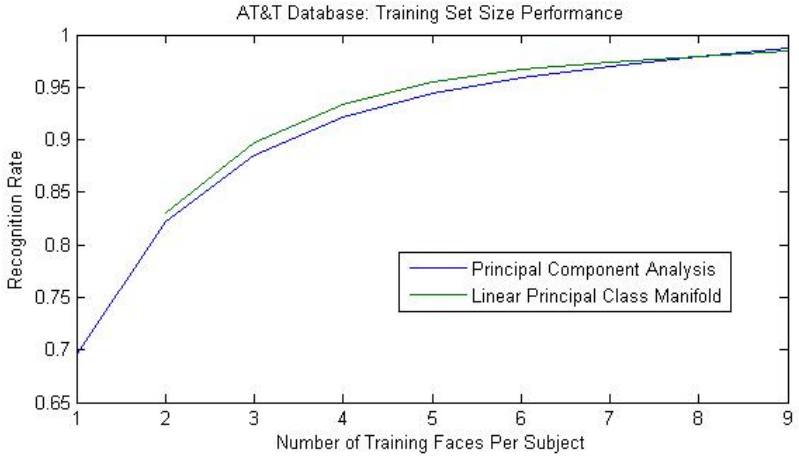

Figure 8: Training Set Size and Recognition

We also observed a relationship between the number of vectors stored and the recognition rates. This is interesting from an engineering perspective as it is a trade off between memory requirements and performance. This test was also performed with Eigenface and Linear Principal Class Manifold implementations with nearest neighbour classification. For Eigenface, the number of vectors stored is essentially the dimension of the Eigenbasis. For Linear Class Manifolds, this is not the case, as the number of vectors stored includes the mean vector for each subject, and the number of Eigenvectors for each subject. The results are displayed in Figure 9.

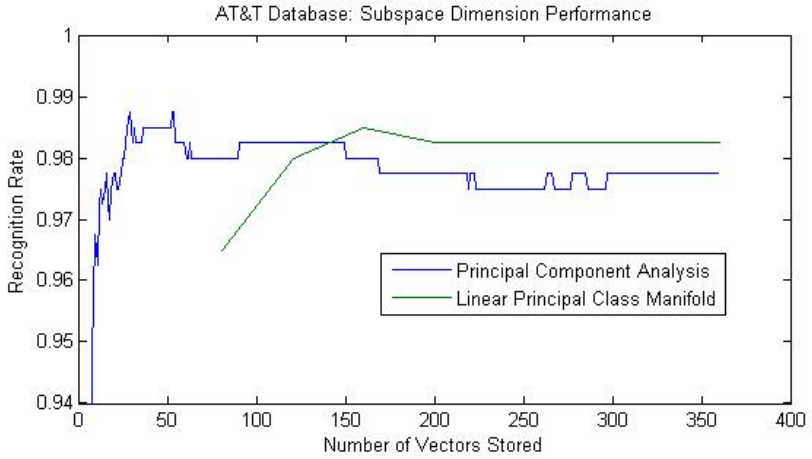

Figure 9: Stored Feature Vectors and Recognition

Finally, we wished to compare the best overall recognition rates of Eigenface, KPCA, Linear Principal Class Manifolds, and Kernel Principal Class Manifolds. The results of these simulations are shown in Table 1.

Method	**Recognition Rate**
Principal Component Analysis	0.9875
Kernel Principal Component Analysis	0.9775
Linear Principal Class Manifold	0.9850
Kernel Principal Class Manifold	0.9850

Table 1: AT&T Database Recognition Rates

4.3 Yale Faces

After testing was performed on the AT&T Database, we wanted to see if similar results would be found on a well established database with more variation. The Yale Face Database is outlined in [4], where it is used to test algorithm

robustness to lighting and other variations. This database is composed of fifteen subjects with eleven 320×243 greyscale images each. A centered and cropped version of this database was used for testing. Sample images are displayed in Figure 10.

Figure 10: Sample of Yale Database Images

We tested the relationship between dimension and recognition rates for all the methods on the Yale database. No plot is included, as the resulting trends closely resembled the results for the AT&T faces, in Figure 7.

Again, we tested on this database to observe the effects of the number of training images of recognition rates. The results of this test for Eigenface and Linear Principal Class Manifold implementations with nearest neighbour classification are show in Figure 11.

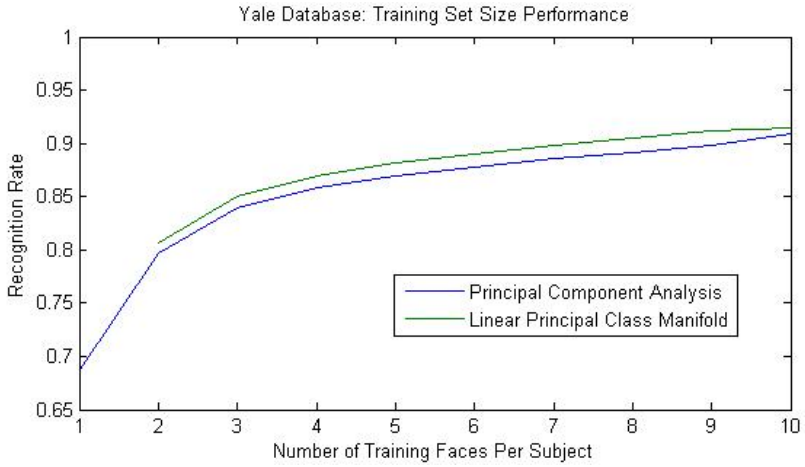

Figure 11: Training Set Size and Recognition

Figure 12 displays the effects of the number of feature vectors kept on recog-

nition rates for Eigenface and Linear Pincipal Class Manifold implementations with nearest neighbour classification.

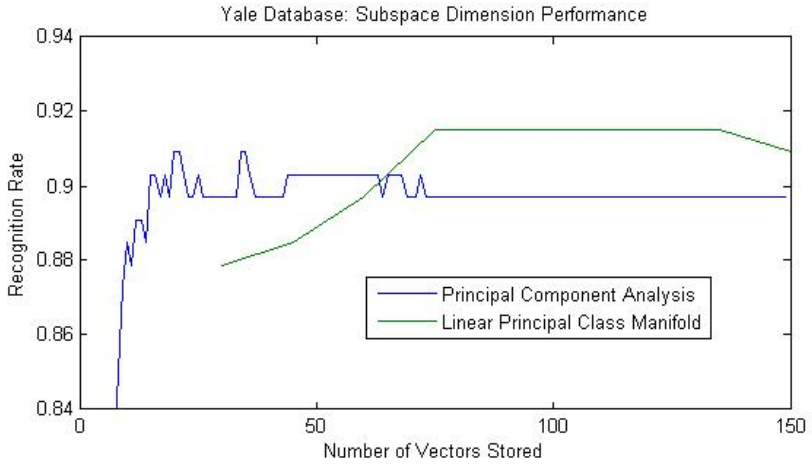

Figure 12: Stored Feature Vectors and Recognition

We also investigated the overall best recognition rates of each method on the Yale Database. These recognition rates are shown in Table 2.

Method	**Recognition Rate**
Principal Component Analysis	0.9091
Kernel Principal Component Analysis	0.9091
Linear Principal Class Manifold	0.9152
Kernel Principal Class Manifold	0.9152

Table 2: Yale Database Recognition Rates

4.4 Facebook

After the success of testing on the Yale and At&T databases, extensive testing was done on the Facebook database that was constructed. As above, a test was run to observe recognition rate as it changes due to the number of feature vectors kept. The test was run on 194 different subjects, using 50 photos as a training set, and using 10 photos as test data. Recognition rates are very low for all methods. Additionally, these results display notable differences in overall recognition rates. Peak recognition rates are shown in Table 3. It should be noted that the difference between the traditional methods and the class manifolds is considerably larger on this database, especially considering the size of the database.

Method	Recognition Rate
Principal Component Analysis	0.0820
Kernel Principal Component Analysis	0.0732
Linear Principal Class Manifold	0.2088
Kernel Principal Class Manifold	0.2093

Table 3: Facebook Database Recognition Rates

5 Discussion

5.1 Test Analysis

In order to fairly evaluate our different methodologies, caution had to be taken against direct comparisons. Particularly between databases, raw numbers are not significant, as the variances between the types of photos are substantial. As well, the differences between the two main types of methods (the traditional approaches and the class manifold methods) means that comparison of dimensionality is difficult. As a result, any comparison can not be purely quantitative, but rather must be carefully considered.

One clear trend that is obvious in Figures 9 and 12 is that the performance of the systems is highly related to the number of feature vectors that are stored. On both databases, the performance quickly rises as the number of vectors stored increases, plateauing fairly early on. This is something that must be considered in the design of the system - deciding how many vectors to keep has an impact on the performance of the system, but it also impacts the computational cost.

Looking at a single database allows us to more easily compare the effectiveness of the various modelling approaches. For instance, looking at the peak identification rates for each database shows that in an optimal case, the class manifold methods are an improvement over the traditional PCA or KPCA approaches. The improvement is slight for AT&T and Yale faces, but is very significant in the case of the Facebook database, suggesting that the class manifold methods are more robust when it comes to variances in the database. While the low overall rates in the Facebook database are meaningful, it is likely that with some further preprocessing, the overall rates could be improved, by reducing the variation within the database. Overall, however, the roughly two-fold increase in the recognition rate in the Facebook database indicates that our class manifold methods are successful.

These comparisons ignore any cost associated with storing data, and it is here that a more careful analysis must be done. We notice that the PCA and KPCA methods require keeping a fixed number of Eigenvectors, related to the size of the database; on the other hand, the class manifold methods require keeping a certain number of vectors per subject. While all methods require

increased storage as the database size increases, the nature of the growth is different. In general, it appears that the class manifold methods do require a larger amount of storage; however, the nature of their classification techniques means that a smaller amount of data must be used at any given time. In comparison, the traditional approaches must keep all the Eigenvectors stored in memory at once during classification. This gives class manifold methods an advantage in dealing with large databases, as they are more easily able to take advantage of secondary storage such as hard drives. As the availability of inexpensive storage has ballooned in recent years, the ability to take advantage of this has become more valuable. From a different perspective, these are all relative considerations. All of the algorithms have modest memory requirements with regard to today's computers.

Another factor that needs to be considered when analyzing the methods is the speed at which they run. Traditional PCA requires large matrix multiplications. KPCA requires many kernel evaluations, computationally comparable to dot products. PCA and KPCA require very similar amounts of computation, with PCA needing slightly less. Classification requires a nearest-neighbour search, but in a lower dimensional subspace. In comparison, the class manifold methods require distance computations to be done in the original image space, so there are a few more operations with large dimension. An advantage of class manifold methods is that the classification approach is highly parallelizable, because the comparison to each subject's subspace is independent of each other subject. Thus the computations can be done in parallel. This lends itself to modern multicore computers, or to programming on modern graphical processing units, which allow computations to be done using several hundred parallel threads.

5.2 Further Considerations

Beyond mere performance results, our investigations do raise some other questions. Clearly class manifold methods perform well and are worth some consideration, especially with regard to the results on the Facebook data. However, we recall that the Gaussian kernel did not work for this method, and we had to use our own exponential kernel: $k(x, y) = e^{(x \cdot y)/\sigma^2}$. At this time, we have not proved this kernel satisfies Mercer's theorem. The fact that it achieved good recognition rates suggests that the kernel evaluations do in some sense represent feature space dot products, a property of kernels satisfying Mercer's theorem. This is worth some more analysis, should this be further pursued.

Another question is whether or not non-linear class manifolds truly offer any benefit over linear class manifolds. Except for our exponential kernel, all other kernels performed worse than linear class manifolds. We recall that when the kernel is simply the dot product, KPCA becomes PCA. Our kernel is a strictly increasing function of the dot product. The fact our kernel's results are *very* close to the linear class results, which use PCA (the dot product reduction of

KPCA), might be simply due to the monotonic kernel inheriting the effectiveness of PCA. On the other hand, the odd degree polynomials are also increasing in the dot product. For these we observed good results, but lower than the former two methods. So whatever way the exponential acts on the dot product may be somehow better for recognition than the polynomials. There are no conclusions at this point. Rather, that the Gaussian kernel failed and that other kernels never made noteworthy performance boosts over the linear class manifold method may suggest limited efficacy of the kernel class manifold method.

5.3 Ethics

A common concern with facial recognition technology is with regard to the ethics of its application. This is particularly important in recent years, as the popularity of social media has made it possible to get pictures and personal information about people without their explicit permission. Even in our system, there were concerns about using photos from Facebook, despite limiting ourselves to publicly available photographs. The fact that users are largely unaware of how their photographs can be used raises the question of whether it is the responsibility of engineers to ensure these photographs are used in ethical ways, or whether the burden is on the users of social media to know that information they upload is publicly available. This gets further complicated when the services being used introduce convoluted privacy policies, which makes it difficult for the average user to understand exactly what rights they are giving when they upload a photograph. As a result, this introduces a possible burden on the service to educate its users as to how their data can be used by outsiders.

While these concerns currently seem to be strictly academic, as facial recognition technology improves, it becomes prudent to consider these factors before issues arise. To continue using Facebook as an example, applications of facial recognition seem to be currently limited to identifying friends in photographs; however, with the large amount of personal data attached to a Facebook identity, it is possible that a service could developed to allow third parties to identify demographic or consumer patterns from photographs they took. A more concrete example would be that of a store using a camera to identify shoppers, finding patterns in the Facebook "likes" of their customers, and tailoring advertising to better target this group. This is likely to make a lot of people uncomfortable. As the technology continues to improve, and as social media becomes more ubiquitous, the need for more serious discussion about this becomes vital.

6 Conclusion

We have successfully implemented a variety of algorithms for facial recognition. After studying and becoming familiar with the existing methods of Principal Component Analysis, or Eigenfaces, and Kernel Principal Component Analysis, we implemented algorithms to perform recognition using them. Then, we mod-

ified the algorithms to generate a principal manifold for each subject, instead of a single principal manifold, and adjusted our classification techniques accordingly. All four methods were then tested on the AT&T and Yale databases, yielding similar peak recognition rates for all of them. We then used the Facebook Graph API to create our own database, with a far greater number of photographs, and a higher variation spread of data. Testing on this, our principal class manifold methods outperformed the traditional methods over double. The success on this test does suggest a robustness to variation on the part of the class manifold methods. We have considered other aspects of the algorithms and their computational and mnemonic requirements. The class manifold methods do have a larger memory requirement, but are still completely feasible with today's technology. In conclusion, the methods are certainly viable and appear to outperform traditional principal manifold methods in certain situations, and thus warrant further investigation. An important track of investigation is into the viability of non-linear class manifolds over linear class manifolds, which as of this point is not confirmed.

A Appendix: Centering in the Feature Space

A.1 Centering for KPCA

In order to perform PCA on a set of training vectors $\{x_1, x_2, \ldots, x_T\}$, we require that these vectors be zero mean. Therefore, when performing KPCA, the training data must be centered in the feature space. That is $\{\psi(x_1), \psi(x_2), \ldots, \psi(x_T)\}$ must have zero mean. In order to do this, we perform KPCA with a new mapping $\tilde{\psi}$ that is a centered version of ψ. $\tilde{\psi}$ satisfies

$$\tilde{\psi}(x_j) = \psi(x_j) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i)$$

The centered mapping $\tilde{\psi}$ is defined implicitly via a kernel function $\tilde{k}$, where $\tilde{k}(x, y) = (\tilde{\psi}(x) \cdot \tilde{\psi}(y))$. Then

$$\begin{aligned}
\tilde{k}(x_n, x_m) &= (\tilde{\psi}(x_n) \cdot \tilde{\psi}(x_m)) \\
&= (\psi(x_n) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i)) \cdot (\psi(x_m) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i)) \\
&= \psi(x_n) \cdot \psi(x_m) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i) \cdot \psi(x_m) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_n) \cdot \psi(x_i) \\
&\quad + \frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T}\psi(x_i) \cdot \psi(x_j) \\
&= k(x_n, x_m) - \frac{1}{T}\sum_{i=1}^{T}k(x_n, x_i) - \frac{1}{T}\sum_{i=1}^{T}k(x_i, x_m) \\
&\quad + \frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T}k(x_i, x_j) \\
&= K_{nm} - \frac{1}{T}\sum_{i=1}^{T}1_{ni}K_{im} - \frac{1}{T}\sum_{i=1}^{T}K_{ni}1_{im} \\
&\quad + \frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T}1_{ni}K_{ij}1_{jm}
\end{aligned}$$

where, K is the matrix of kernel evaluations. In matrix form, we can compute the matrix of centered kernel evaluations $\tilde{K}$ as

$$\tilde{K} = K - 1_T K - K 1_T - 1_T K 1_T$$

where 1_T is the $T \times T$ matrix with entries $(1_T)_{ij} = \frac{1}{T}$. This centers the training data in the feature space.

We also must center the projections of a face onto the feature space training vectors. Suppose we have a test face x. Then if K^{test} is its vector of kernel evaluations with the training faces, $K_i^{test} = k(x, x_i)$. If $\tilde{K}^{test}$ is the centered version of this vector,

$$\begin{aligned}\tilde{K}_n^{test} &= (\tilde{\psi}(x) \cdot \tilde{\psi}(x_n)) \\ &= (\psi(x) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i)) \cdot (\psi(x_n) - \frac{1}{T}\sum_{i=1}^{T}\psi(x_i))\end{aligned}$$

Following a very similar derivation, we find the following equation for centering projections:

$$\tilde{K}^{test} = K^{test} - 1_T^{'}K - K^{test}1_T + 1_T^{'}K1_T$$

where $1_T^{'}$ is the T-component vector with all entries equal to $\frac{1}{T}$. Using this formula and the above, we are able to center our training data and our projections in the feature space, making the zero mean assumption in the KPCA derivation feasible.

A.2 Centering for Kernel Principal Class Manifolds

When computing distances to principal manifolds in the feature spaces of people, we must first center the new image x in these feature spaces first. We assume that the training data for each person, $\{x_{i1}, x_{i2}, \ldots, x_{iT}\}$, has been centered in the feature space as outlined in Appendix A.1. Then person i has a centered mapping $\tilde{\psi}_i$ implicitly defined by the kernel function $\tilde{k}_i$, $i \in \{1, 2, \ldots, P\}$. In order to compute the distance to the hyperplane in the feature space, d_i, we use this centered mapping.

$$\begin{aligned}d_i &= (\tilde{\psi}_i(x) \cdot \tilde{\psi}_i(x)) - \sum_{j=1}^{M}(\tilde{\psi}_i(x) \cdot \tilde{V}^{ij}) \\ &= ((\psi_i(x) - \frac{1}{T}\sum_{m=1}^{T}\psi_i(x_{im})) \cdot (\psi_i(x) - \frac{1}{T}\sum_{m=1}^{T}\psi_i(x_{im})) \\ &\quad - \sum_{j=1}^{M}((\psi_i(x) - \frac{1}{T}\sum_{m=1}^{T}\psi_i(x_{im})) \cdot \tilde{V}^{ij}) \\ &= k_i(x,x) - \frac{2}{T}\sum_{m=1}^{T}k_i(x, x_{im}) + \frac{1}{T^2}\sum_{m=1}^{T}\sum_{n=1}^{T}k_i(x_{im}, x_{in}) \\ &\quad - \sum_{j=1}^{M}((\psi_i(x) - \frac{1}{T}\sum_{m=1}^{T}\psi_i(x_{im})) \cdot \tilde{V}^{ij})\end{aligned}$$

$$
\begin{aligned}
&= k_i(x,x) - \frac{2}{T}\sum_{m=1}^{T} k_i(x,x_{im}) + \frac{1}{T^2}\sum_{m=1}^{T}\sum_{n=1}^{T} k_i(x_{im},x_{in}) \\
&\quad - \sum_{j=1}^{M} \psi_i(x)\cdot \sum_{k=1}^{T} \tilde{\alpha}_k^{ij}\tilde{\psi}_i(x_{ik}) + \frac{1}{T}\sum_{j=1}^{M}\sum_{m=1}^{T} \psi_i(x_{im})\cdot \sum_{k=1}^{T} \tilde{\alpha}_k^{ij}\tilde{\psi}_i(x_{ik}) \\
&= k_i(x,x) - \frac{2}{T}\sum_{m=1}^{T} k_i(x,x_{im}) + \frac{1}{T^2}\sum_{m=1}^{T}\sum_{n=1}^{T} k_i(x_{im},x_{in}) \\
&\quad - \sum_{j=1}^{M}\sum_{k=1}^{T} \tilde{\alpha}_k^{ij}\psi_i(x)\cdot(\psi_i(x_{ik}) - \frac{1}{T}\sum_{l=1}^{T}\psi_i(x_{il})) \\
&\quad + \frac{1}{T}\sum_{j=1}^{M}\sum_{m=1}^{T}\sum_{k=1}^{T} \tilde{\alpha}_k^{ij}\psi_i(x_{im})\cdot(\psi_i(x_{ik})) - \frac{1}{T}\sum_{l=1}^{T}\psi_i(x_{il})) \\
&= k_i(x,x) - \frac{2}{T}\sum_{m=1}^{T} k_i(x,x_{im}) + \frac{1}{T^2}\sum_{m=1}^{T}\sum_{n=1}^{T} k_i(x_{im},x_{in}) \\
&\quad - \sum_{j=1}^{M}\sum_{k=1}^{T} \tilde{\alpha}_k^{ij} k_i(x,x_{ik}) + \frac{1}{T}\sum_{j=1}^{M}\sum_{k=1}^{T}\sum_{l=1}^{T} \tilde{\alpha}_k^{ij} k_i(x,x_{il}) \\
&\quad + \frac{1}{T}\sum_{j=1}^{M}\sum_{m=1}^{T}\sum_{k=1}^{T} \tilde{\alpha}_k^{ij} k_i(x_{im},x_{ik}) - \frac{1}{T^2}\sum_{j=1}^{M}\sum_{m=1}^{T}\sum_{k=1}^{T}\sum_{l=1}^{T} \tilde{\alpha}_k^{ij} k_i(x_{im},x_{il})
\end{aligned}
$$

Recall that $\left\{\tilde{\alpha}^{i1}, \tilde{\alpha}^{i2}, \ldots, \tilde{\alpha}^{iT}\right\}$ are the Eigenvectors of $\tilde{K}_i$, the centered kernel matrix of person i.

B Appendix: Software Implementation

We have included the relevant code for our project on a CD-ROM.

References

[1] FBI, "FBI Biometric Center of Excellence," 2011. [Online]. Available: http://www.fbi.gov/about-us/cjis/fingerprints_biometrics/biometric-center-of-excellence/files/face-recognition.pdf. [Accessed 2 April 2014].

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

[3] Ferdinando Samaria, Andy Harter. "Parameterisation of a Stochastic Model for Human Face Identification," *Proceedings of 2nd IEEE Workshop on Applications of Computer Vision*, Sarasota FL, December 1994.

[4] P. N. Bellhumer, J. Hespanha, and D. Kriegman. "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Special Issue on Face Recognition, 17(7):711–720, 1997.

[5] B. Schölkopf, A. Smola, and K.-R. Muller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation*, vol. 10, no. 5, pp. 1299-1319, 1998.

[6] B. Moghaddam, "Principal Manifolds and Probablistic Subspaces for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 24., no. 6, pp. 780-788, 2002.

[7] N. Poulad, A. Cerisano, N. Cimolai, "Kernel Principal Component Analysis for Facial Recognition," BSc (Eng) Thesis, Dept. Math & Stats, Queen's University, Kingston, ON, 2011.

[8] L. Devroye *et al.*, "Nearest Neighbour Rules," in *A Probabalistic Theory of Pattern Recognition*, 1st ed., New York, Springer, 1996, ch. 5.