ELECTROENCHEPALOGRAPHY PRE-PROCESSING AND CLASSIFICATION TECHNIQUES

KELLIE WUTZKE

Abstract. In this paper, we construct algorithms designed to make the classification of electroencephalography (EEG) signals more accurate and efficient. These techniques include probabilistic analysis, singular value decomposition, and visualization techniques involving cluster analysis. Using these methods, we analyze EEG data from ten subjects performing five different mental tasks. The data is separated into windows of space and time, modified through Singular Value Decomposition (SVD), and entered into a classifier constructed in Matlab. The tasks were classified with 96.6% overall accuracy, with task 4 being the least likely task to be classified mistakenly as another task. Tasks 1 and 2 were most likely to be confused for each other.

Contents

Introduction

The human brain is an intricate machine. Each thought creates a series of electrical interactions between neurons, spreading currents over different regions of the brain. Given the correct equipment, it is possible to measure the change in these electrical patterns over time (via electroencephalography, or EEG), albeit with poor spatial resolution. Previously, this information was used mostly to look for anomalies associated with epilepsy and other disorders; in this paper, however, we explore the use of electrical brain data and the mathematical concepts of classification as a means of separating distinct mental tasks. The ability to accurately recognize different thoughts, once refined, has many practical applications. For example, these techniques may ultimately provide a method for quadriplegics to perform simple tasks on a computer with mental commands [1]. Emotive Systems has created a video game which is connected to the user via an EEG cap, allowing real-time thoughts to translate directly into on-screen action. Currently, the game can detect basic emotions via facial expressions, states of either calm or excitement, and conscious intentions to spatially move an object on the screen [2].

Ultimately, in section 8.1, we analyze EEG data taken from 5 subjects performing 10 trials each of 5 mental tasks. First, though, we begin by approaching the subject of a classifier in broader terms. For example, one may choose to perform analysis on the data to extract certain features, then build a simple classifier; or, one may do no analysis on the data and build a more complex classifier. In order to understand data classification, we begin with the former, and the "simple classifier" used in this example will be the Bayesian classifier. The Bayesian classifier is commonly used and works effectively when data can be clustered into groups according to probabilities. When the data is more complicated, we must extract important features with which we can better define our classes. Singular Value Decomposition (SVD) provides a quick and convenient method for extracting features and greatly reduces the volume of data to analyze. Furthermore, SVD will return our data in the form of a matrix which contains basis vectors that maximize the variance between different classes of data (i.e. the signals of one mental task compared to those of a different mental task), a concept called Principal Component Analysis (PCA) [3]. We use Matlab to aid in the intensive calculations involved when performing SVD and running the classifier algorithm on our data.

Our EEG classifier resembles that of Anderson et al [3]. The data initially forms 50 6x2500 matrices for each subject's EEG scan of a specific task during a specific trial (6 channels by 2500 seconds)[1] . Analyzing this data unmodified would result in a spatial picture of each mental task. Unfortunately, the spatial resolution of the EEG is poor and the mental tasks performed are complicated, causing many regions of the brain to activate. This makes separating tasks virtually indistinguishable as a function of space. Instead, we break apart our EEG matrices to represent space and time. Using what they call short-time PCA, Anderson et al found that the tasks were classified between 67.5% to 87.7% correctly, with task 2 being the least likely to be represented as a different class. Tasks 3 and 1 and tasks 4 and 5 were confused with each other fairly often [3]. Here, we imitate these results with the

[1]There are five tasks total. Task 1 is rest. Task 2 is visualization of drawing a letter. Task 3 is multiplying two three-digit numbers together. Task 4 is mentally rotating an object. Task 5 is counting to self.

classifier we built, attempt to improve the classification error, and then perform statistical analysis on the classifier's effectiveness.

1. Classifiers: An Overview

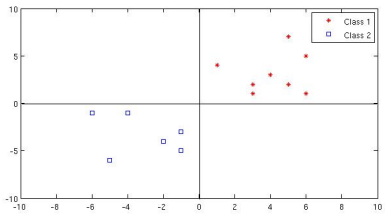

Figure 1: Random points which can be separated linearly.

1.1. **The General Purpose of Classifiers.** A classifier is an algorithm whose input is a set of data objects and whose output is an index which indicates to which class each data object belongs. A classifier consists of a preprocessor, which reads a stream of incoming data and transforms it into a vector or matrix, and a feature extractor, which takes either the pre or post-transduced data and removes as much noise as possible from the signal. The feature extractor may also define for the classifier some strong characteristics in the data, aiding in pattern identification [4]. Several problems occur when classifying data. First, the data needs to be segmented prior to transduction (envision a machine which counts apples on a conveyor belt). Furthermore, we require that the machine be able to distinguish one object from the next (where one apple ends and the next begins)[2]. If two apples are on top of or right next to each other then error is introduced when they are misclassified or miscounted. Second, if the features chosen are not distinct enough (choosing burgundy vs. cardinal red apples as opposed to red vs. green) then the classifier will be more likely to classify one as the other. Also, having too few features can result in poor classification. Having one feature gives the classifier one parameter for defining each object; using several features offers multiple defining characteristics for each data object. Thus, as long

[2]This analogy is not far from our situation; we can imagine our EEG data as heavily sampled but discrete signals which fire at a near continuous rate.

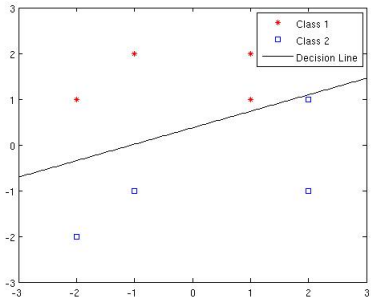

Figure 2: Given these points, we can find a line which classifies the data into two sets.

as the majority of the features are determined correctly, the data object will not be misclassified.

There are many ways in which to classify a set of data. The methods discussed below are linear, exact, and Bayesian classification. While simple, they demonstrate nicely the concepts and problems that arise in creating classifiers.

1.2. **Linear.** Let us consider a simple problem of classification. Say we are given a graph of points (Figure 1) in $\mathbb{R}^2$ which seem to have some reasonable separation between them. These points are considered our training data–each one is already labeled with their respective class. In this example, we create two classes, where Class 1, denoted C_1, is made of points $(x_1, y_1) \dots (x_n, y_n)$ and Class 2, denoted C_2, consists of points $(x_{n+1}, y_{n+1}) \dots (x_{n+m}, y_{n+m})$ (see Figure 1). The classifier in this example is considered linear because we find the line that gives a good separation between the classes. We can express this using linear algebra by finding a line of the form $f(x_i, y_i) = ax_i + by_i + c$ where $i = 1, 2, \dots, n$ for C_1 and $i = n+1, n+2, \dots, n+m$ for C_2. Class C_1 contains points in the 3rd quadrant and C_2 contains points in the 1st quadrant. To find a, b, and c for our classification we write a series of equations, setting them equal to two arbitrary points for C_1 and C_2 which are relatively near the grouping of points and preferably equidistant from the origin (in the case of Figure 1). These are our *decision points*. Let's choose 1 for C_1 and -1 for C_2. Our system of equations becomes:

$$\begin{aligned} ax_1 + by_1 + c &= 1 \\ ax_2 + by_2 + c &= 1 \\ &\vdots \\ ax_n + by_n + c &= 1 \\ \hline ax_{n+1} + by_{n+1} + c &= -1 \\ ax_{n+2} + by_{n+2} + c &= -1 \\ &\vdots \\ ax_{n+m} + by_{n+m} + c &= -1. \end{aligned}$$

In matrix form, these equations become:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & & \\ x_{n+1} & y_{n+1} & 1 \\ x_{n+m} & y_{n+m} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ -1 \\ -1 \end{bmatrix}.$$

We now solve this equation for the variables a, b, and c. The line $ax_i + by_i + c = 0$ is our decision function and will be equidistant from -1 and 1, where -1 and 1 are weights that are chosen *ad hoc* to label some points negative and some positive. This aids in their classification according to our decision line. We do not expect an exact solution because we are solving for three variables with 30 unknowns. Rather, we want to find a separation line that best fits between our two classes. In linear

algebra, we think of this problem in terms of the normal equations corresponding to

$$A^T A\mathbf{x} = A^T\mathbf{b}.$$

In Matlab, solving this system of unknowns simply returns the line of best fit according to the least-squared distances between the data points and the decision line. Because our system has full rank and $A^T A$ is invertible, our solution is unique.

Use this model to analyze a real data set. Given the set of points plotted in Figure 2, we can write the following matrix equation:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & -1 & 1 \\ 2 & 1 & 1 \\ -2 & 1 & 1 \\ -1 & 1 & 1 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}.$$

Using Matlab, we solve for a, b and c to find $a = -0.2247, b = 0.6235$, and $c = -0.2338$. We can then draw the line of best fit from the resulting equation, $y = \frac{-0.2447x + -0.2338}{0.6325}$. This is the line shown in Figure 2. The points above the line are contained in Class 1, and below in Class 2. It is easy to see that in some cases the line may incorrectly classify some of the training data points (in this example, however, the line separates both classes without error). See Appendix A.1 for the code described above and the output.

1.3. **Exact.** Using the same data shown in Figure 1, we instead create a classifier which separates each set of data exactly, meaning each point in the training data is on the side of the line which corresponds to its proper class. If we do, we arrive at something like Figure 3. It is obvious that we should never use this type of classification. Of course, the error when using the training data is zero, which seems ideal. However, problems arise when new data is input. If we add points to our graph, many near the exact classifer's decision line that would be classified as Class 2 in Figure 2 are now classified as Class 1. Because the function to separate each class is so exact, the probability that new points will be misclassified is high. However, between the linear classifier and the exact, a subtle point can be made: A classifier which is either too general or too specific will not be very successful. In order to minimize error for

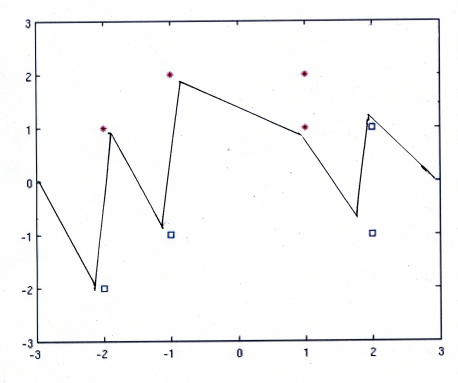

Figure 3: An exact classification. Note that while this classification is perfect for the set of data input used to create it, it will have a high error when new data is classified.

both the training data and novel data, some middle ground between the two must be found.

1.4. **Bayesian.** If, instead, we imagine our data vectors as forming density functions in high dimensional space, then we may still solve the same general problem of dividing up our region of space into classes, but we require more complicated pattern classification techniques. If the distribution of the random data vectors is given, then the classifier which best minimizes classification error is the Bayes classifier [5]. For simplification, assume our data forms two-dimensional density functions which can be assigned to one of two classes, ω_1 or ω_2.

We know based on the initial data (the training set) that there are *a priori* probabilities $P(\omega_1)$ and $P(\omega_2)$ which represent the likelihood that the next data input, x, is in either class. Thus, the greater the disparity between the class probabilities, the better our guess for x will be. In other words:

$$\text{If } P(\omega_1) \gg P(\omega_2) \text{ choose } \omega_1 \text{ and vice versa.}$$

We denote the density probability function of x for the class ω_i as $p(x \mid \omega_i)$ and the difference in the two classes' probability density functions as $p(x \mid \omega_1) - p(x \mid \omega_2)$. A larger separation between the density functions indicates a well defined feature (green apples are not as likely to be classified as red apples then pink ones).

For two classes, Bayes Formula states:

$$P(\omega_i \mid x) = \frac{p(x \mid \omega_i)P(\omega_i)}{p(x)} \tag{1}$$

where

$$p(x) = \sum_{i=1}^{2} p(x \mid \omega_i)P(\omega_i). \tag{2}$$

Bayes formula tells us that the probability of a class ω_i being chosen given some new data, x, is equivalent to the probability density function of the class based on the known data multiplied by the probability of that class. This is scaled by an evidence factor, $p(x)$, which represents the frequency with which a pattern with feature value x will actually be measured. Its existence is unimportant except to assure that $P(\omega_1 \mid x) + P(\omega_2 \mid x) = 1$ [4]. Ultimately, our goal is to minimize the probability of error given x, or $P(\text{error} \mid x)$. It follows then that when $P(\omega_1 \mid x) > P(\omega_2 \mid x)$ we should choose ω_1 and vice versa (this is called *Bayes decision*). To justify this argument, we consider the average probability of error:

$$P(\text{error}) = \int_{-\infty}^{\infty} P(\text{error}, x)dx = \int_{-\infty}^{\infty} P(\text{error} \mid x)p(x)dx.$$

This integral is simply a substitution using the principle shown in Equation 2. If we minimize $P(\text{error} \mid x)$, we minimize the integral and justify Bayes decision. We can substitute Equation 1 into *Bayes decision* and eliminate the evidence factor, $p(x)$, to get the following decision rule:

$$\text{If } p(x \mid \omega_1)P(\omega_1) > p(x \mid \omega_2)P(\omega_2), \text{ choose } \omega_1.$$

Two special cases provide more insight into Bayesian decision theory. If $P(\omega_1) = P(\omega_2)$ then x is equally likely to be in either class and our decision is based entirely on the likelihoods $p(x \mid \omega_i)$. If instead $p(x \mid \omega_1) = p(x \mid \omega_2)$, then we base our decision entirely on the prior probabilities, $P(\omega_i)$ [4].

We explore the uses and application of the Bayesian classifier further in the following section, specifically when our probability density functions are Gaussian.

2. Bayesian Decision Theory: Univariate and Multivariate Densities

In this section, we summarize material as found in Duda [4]. Now that we have determined that the Bayes classifier depends on the conditional probabilities, $p(x \mid \omega_i)$, we look at the Gaussian density function whose feature vectors $\mathbf{x}$ for any class ω_i are randomly distributed and continuously valued. Before we begin, note that the expected value of a function, $f(x)$, can be written:

$$\mathrm{E}[f(x)] \equiv \int_{-\infty}^{\infty} f(x)p(x)dx$$

where $p(x)$ is the probability distribution function of x. If the values of x are restricted to a discrete set, $\mathcal{D}$, then our expected value is the sum

$$\mathrm{E}[f(x)] = \sum_{x \in \mathcal{D}} f(x)P(x),$$

where $P(x)$ is the probability mass[3] at x.

2.1. Discriminate Functions.

Remembering that for classes ω_i,

$$P(\omega_i \mid \mathbf{x}) = \frac{P(\omega_i)p(\mathbf{x} \mid \omega_i)}{p(\mathbf{x})},$$

we can approximate $P(\omega_i \mid x)$ for each class where $i = 1, 2 \ldots n$ and n is the total number of classes. If we call this classification k, we want to find the maximum probability (where $p(x) \neq 0$),

$$k = \arg\max_i \left\{ \frac{P(\omega_1)p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x})}, \frac{P(\omega_2)p(\mathbf{x} \mid \omega_2)}{p(\mathbf{x})}, \ldots, \frac{P(\omega_n)p(\mathbf{x} \mid \omega_n)}{p(\mathbf{x})} \right\}.$$

Because we are simply concerned with the index corresponding to the maximum and not the value we can multiply each element in the set by $p(x)$ to get

$$k = \arg\max_i \{P(\omega_1)p(\mathbf{x} \mid \omega_1), P(\omega_2)p(\mathbf{x} \mid \omega_2), \ldots, P(\omega_n)p(\mathbf{x} \mid \omega_n)\}.$$

If the value of the index corresponding to the maximum is not unique, we can randomly assign x to one of the indices with a maximum value. One way to represent pattern classifiers is with a set of discriminant functions. If we call our set of discriminant functions $g_i(\mathbf{x})$ for $i = 1, 2, \ldots, n$ then the feature vector $\mathbf{x}$ is assigned to a class ω_i with the largest discriminant if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \text{ for all } j \neq i.$$

We can now take $g_i(\mathbf{x}) = P(\omega_i)p(\mathbf{x} \mid \omega_i)$ so that the maximum discriminant function is equal to the maximum probability. Because we are only concerned with maximum values, taking $f(g_i(\mathbf{x}))$, where $f(\cdot)$ is a monotonically increasing function, will not affect our classification. One such function is $\ln(x)$. Thus we can say

$$f(g_i(\mathbf{x})) = \ln(p(x \mid \omega_i)P(\omega_i))$$

and furthermore,

$$f(g_i)(\mathbf{x}) = \ln(p(x \mid \omega_i)) + \ln(P(\omega_i)). \tag{3}$$

[3]The probability mass function $P(x)$ describes the probability for each value of the random variable, x [7].

We now have a discriminant function which can calculate values using the same index, i, as the arg max function mentioned earlier, and return a maximum value (though the maximum values will not necessarily be equal).

2.2. **The Univariate Case.** If we assume that the unknown distribution $p(x)$ is a univariate distribution such that

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{4}$$

then the expected value of x is

$$\mu \equiv \mathcal{E}(x) = \int_{-\infty}^{\infty} xp(x)dx$$

with variance

$$\sigma^2 \equiv \mathcal{E}[(x-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx.$$

[4] According to the *Central Limit Theorem*, data that has undergone many small, random effects is approximately normally distributed [6]. Since many patterns in the natural world are ideal models which have undergone many small, random perturbations, the Gaussian typically replicates the actual probability distribution of these patterns nicely [4].

We can use graphing to help us visualize this two-dimensional problem with two classes. Using Equation 4, we get

$$p(x \mid \omega_1) = \frac{1}{\sqrt{2\pi}\sigma_1} e^{-(\frac{(x-\mu_1)^2}{2\sigma_1^2})} \tag{5}$$

$$p(x \mid \omega_2) = \frac{2}{\sqrt{2\pi}\sigma_2} e^{-(\frac{(x-\mu_2)^2}{2\sigma_2^2})} \tag{6}$$

where Equation 5 represents Class 1 and Equation 6 represents Class 2. Our discriminant function becomes:

$$g_i(x) = \frac{(x-\mu_i)^2}{2\sigma_i^2} - log(\sigma_i) \text{ for } i = 1, 2. \tag{7}$$

Assume that for some set of training data, $\mu_1 = 1$, $\mu_2 = 7$, $\sigma_1 = 1$ and $\sigma_2 = 3$. Using the Gaussian distribution above, we use Matlab's random number generator and these values of μ and σ to create 500 random numbers for each class (our "training data"). Because the Gaussian distribution is based on probabilities with the μ and σ parameters defining a specific distribution, Matlab can simply generate random numbers based off of the Gaussian probabilities. Next we create a 1000x1 matrix of our data, matrix X, and insert our matrix into g_1 and g_2, which represent the discriminant Gaussian functions pertaining to their respective μ_i and σ_i. We then create a new matrix, G, with the training data where $G = [g_1\ g_2]$. G is a 1000x2 matrix with the values of x for g_1 in one column and the values of x for g_2 in the second. The code

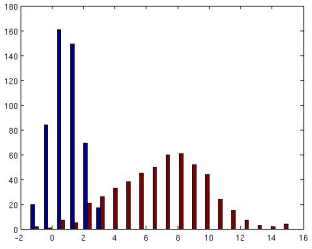

Figure 4: A histogram of data points in classes 1 (blue) and 2 (red).

in Appendix A.2 shows this classifying process, along with a command (on the last line) that compares the values in column one and column two, picks the largest of the two, and then assigns that x to that class (where column one is class one and column two is class two). This data is stored in the new matrix "class".

To compare the above method to one where the class is chosen based on the minimum distance from μ_i to x we use the code given in Appendix A.2, which performs the same operations as above but with columns which contain the absolute value of $\mu_i - x$. The minimum distance is calculated for each point and the corresponding class (Y1 or Y2 in the code) is stored in a new matrix called Z.

For a graph demonstrating the results using both classification methods, see Figure 5. Essentially, we have used the training data to create a classifier which hopefully is not too specific (classifying the training data with no error) but has minimal classification error. In Figure 5, if all of the points in each class can be separated without any misclassification, we expect to see a line from 0 to 500 on the x-axis at $y = 1$ and a second line from 501 to 1000 at $y = 2$. In this example, we see errors of 6% and 8% for our maximum probability classification and minimum distance classification, respectively. Intuitively, we might assume that there are instances when the minimum distance method performs better. For instance, if the data shown in Figure 4 were completely separated, then we might assume a new point would be well classified if we placed it in the class whose mean value is closest. However, we must remember that mean distance classification determines a data point's class based on half the distance between the means, or $\frac{(x-\mu_i)}{2}$. Thus, if the variances are not equal between the classes, the median between the classes' means will favor one class over the other. In fact, this means that the minimum distance classification can only be as good as but never better than the maximum probability classification, and it is only equal when the variances of each class are equal. If our σ_is are equal, Equation 7 becomes:

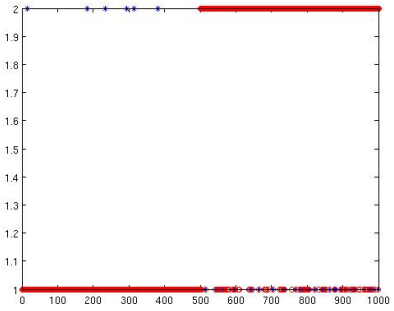

Figure 5: Blue lines and circles represent all x_i chosen for class 1. Pink lines and circles represent all x_i chosen for class 2.

$$\begin{aligned} g_i(x) &= \frac{(x-\mu_i)^2}{2(1)^2} - log(1) \\ &= \frac{(x-\mu_i)^2}{2}. \end{aligned}$$

The minimum of this function will also be the minimum of $\frac{(x-\mu_i)}{2}$. We don't normally expect or observe equal variances between classes. If, in a given scenario, we do find them to be equal then the minimum distance classification is simpler to perform and will not increase our error.

2.3. **The Multivariate Case.** If we assume that the unknown distribution $p(x)$ is a multivariate d-dimensional distribution such that

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}} \mid \mathbf{\Sigma} \mid^{-\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)},$$

then we substitute $p(\mathbf{x} \mid \omega_i)$[4] into Equation 3 to get

$$g_i(\mathbf{x}) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln \mid \Sigma_i \mid + \ln(P(\omega_i)).$$

When plotted in 2-dimensions, these graphs resemble Figure 6. In Figure 6, the regions where the two functions collide is where error will be introduced during classification as this is where it is most likely points in one class will be classified as belonging to the other class.

3. Singular Value Decomposition

Because of the volume and complexity of EEG data, several modifications are required. Initially, our data is stored as a 5x5x10 cell which must be separated into windows of time (for each task, trial, and subject). We discuss our method for doing this in Section 5.2. Splitting our data cell results in 50 6x2500 matrices[5]. To analyze a particular matrix, we begin by first performing Singular Value Decomposition (SVD). The SVD (Singular Value Decomposition) is the primary tool we will use for the analysis, so in this section we will develop the decomposition and look at its properties. We begin with a discussion of The Spectral Theorem and singular values and then discuss how to construct matrices under SVD. Finally, we show that the SVD provides the best basis for a given data set.

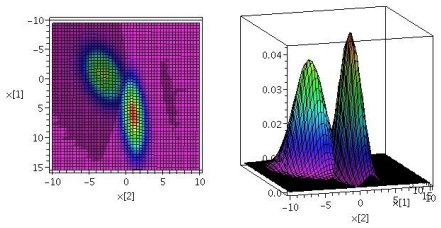

Figure 6: 2-dimensional Gaussian functions representing two separate classes.

3.1. **The Spectral Theorem and Singular Values.** Recall that a matrix, A, is symmetric if $A = A^T$.

Theorem 1. (The Spectral Theorem for Symmetric Matrices [8]) *An $n \times n$ symmetric matrix A has the following properties:*

a. *A has n real eigenvalues, counting multiplicities.*
b. *The dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.*
c. *The eigenspaces are mutually orthogonal, in the sense that eigenvectors corresponding to different eigenvalues are orthogonal.*
d. *A is orthogonally diagonalizable.*

[4]In this d-dimensional Gaussian Σ is a covariance matrix with size d-by-d with an inverse Σ^{-1} and determinant $\mid \Sigma \mid$. Further, $\mathbf{x}$ is d-component column vector, μ is a d-component vector of means, and $(\mathbf{x}-\mu)^T$ is the transpose of $\mathbf{x}-\mu$.

[5]Each 6x2500 array represents six electrodes, each taking 2500 samples in time- Therefore, one matrix represents one EEG recording episode for a single mental task and with one subject.

Given an arbitrary $m \times n$ matrix A, a symmetric matrix B can be constructed from it by taking:

$$B^{n\times n} = A^T A \quad \text{or} \quad B^{m\times m} = AA^T.$$

Now matrix B has a complete set of eigenvalues. Taking $A^T A$, we can compute the set of eigenvalues $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of $B^{n\times n}$ and subsequently an orthonormal basis in $\mathbb{R}^n$, $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\}$. Furthermore, we can show that the following is true:

Lemma 1. *: Let A be an $m \times n$ matrix, and let $B = A^T A$. Then the eigenvalues of B are all non-negative.*

Proof. For $1 \leq i \leq n$, where $\mathbf{v_i}$ is an eigenvector of $A^T A$ and λ_i is its associated eigenvalue:

$$\begin{aligned} \| A\mathbf{v_i} \| &= (A\mathbf{v_i})^T A\mathbf{v_i} = \mathbf{v_i}^T A^T A\mathbf{v_i} \\ &= \mathbf{v_i}^T(\lambda_i \mathbf{v_i}) \\ &= \lambda_i \mathbf{v_i}^T \mathbf{v_i} \\ &= \lambda_i \end{aligned}$$

□

Knowing that the values of λ_i are all nonnegative, we define the singular values of A as the square roots of the eigenvalues of $A^T A$, denoted σ_i for $1 \leq i \leq k$, where k is the minimum of n and m.

Note that that there are m eigenvalues of AA^T, $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ and that the nonzero eigenvalues of AA^T are the same in number and value as the eigenvalues of $A^T A$. Also, if there are r non-zero singular values of A and $\{v_1, v_2, \ldots, v_r\}$ are the associated eigenvectors of A, then $\{A\mathbf{v_1}, A\mathbf{v_2}, \ldots, A\mathbf{v_r}\}$ is an orthogonal basis for $\text{Col}(A)$ and rank $A = r$ [**?**].

3.2. **Performing SVD On a Matrix.** Now we have the necessary tools to perform the SVD. Given the matrix A with the properties described above (A is $m \times n$ with r singular values), we can write

$$A = U\Sigma V^T$$

where Σ is the $n \times m$ diagonal matrix of the singular values of A, V is the $m \times m$ eigenvector matrix for $A^T A$, and U is the $n \times n$ eigenvector matrix for AA^T. Finding Σ is simple–the eigenvalues of $A^T A$ are ordered down the diagonal of Σ from greatest to least value. Zeroes fill in the columns and rows until Σ is the same size as A ($n \times m$). To find V, we find the unit eigenvectors of $A^T A$ and arrange them such that the corresponding eigenvalues are ordered from greatest to least value. If $\lambda = 0$, we include that eigenvector in our matrix. The same procedure is used to find U, but with the matrix AA^T. Alternatively, to find the eigenvectors of U we may calculate $A\mathbf{v_1}$ through $A\mathbf{v_r}$ (where $Av_i = u_i$) and make those into unit vectors. Any columns of U which are not filled by these vectors are found by taking orthogonal unit vectors to u_i. In other words, we take $u_j^T \mathbf{x} = 0$ for $1 \leq j \leq r$ and find a basis for the solution set with the resulting equations. We call the unit eigenvectors of V the *right singular vectors* and the unit eigenvectors of U the *left singular vectors* [8].

Here, we work through an example of SVD:

Suppose

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

First, we find $A^T A$:

$$A^T A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

The eigenvalues of this matrix are 3 and 1. Thus, $\sigma_1 = \sqrt{3}$ and $\sigma_2 = \sqrt{1} = 1$. The corresponding eigenvectors are:

$$\mathbf{v_1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and

$$\mathbf{v_2} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

The unit eigenvectors form the columns of V:

$$V = \frac{1}{\sqrt{2}}[\mathbf{v_1} \qquad \mathbf{v_2}] = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

To construct U, first find $A\mathbf{v_1}$, denoted $(\mathbf{u_1})$, and $A\mathbf{v_2}$, denoted $(\mathbf{u_2})$:

$$\mathbf{u_1} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

and

$$\mathbf{u_2} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}.$$

Taking $u_1^T \mathbf{x} = 0$ and $u_2^T \mathbf{x} = 0$ gives us the relations $x_1 = x_3$ and $x_2 = -x_3$ and thus the vector

$$\mathbf{w}^* = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}.$$

Now we can find $U = \{\mathbf{g_1} \qquad \mathbf{g_2} \qquad \mathbf{w}\}$, where g_1, g_2 and w are the unit eigenvectors for u_1, u_2 and w^* respectively:

$$U = \begin{bmatrix} \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix}.$$

Since our eigenvalues are $\sqrt{3}$ and 1,

$$\Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

and our singular value decomposition is:

$$A = U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

In this problem, we are mapping from $\mathbf{x} \epsilon \mathbb{R}^n$ to $A\mathbf{x} \epsilon \mathbb{R}^m$, or in other words from Row(A) to Col(A^T). If there are r non-zero singular values of A, the dimension of both these spaces is r where $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r}\}$ form a basis for Row(A) and $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_r}\}$ form a basis for Col(A^T). The spaces Null(A) and Null(A^T) may not be equal since the dimension of Null(A) is $n-k$ and the dimension of Null(A^T) is $m-k$. However, this is irrelevant because we disregard the null space and are only concerned with the reduced singular value decomposition.

3.3. **The Karhunen-Loéve Expansion.** The Karhunen-Loéve (KL) Expansion confirms that the SVD provides the best basis of feature vectors[6]. Given a set of p data points, $\{\mathbf{x}^{(1)} \ldots \mathbf{x}^{(p)}\}$, where each $\mathbf{x}^{(i)} \varepsilon \mathbb{R}^n$, we denote the collection of data as a $p \times n$ matrix X such that:

$$X = [\mathbf{x}^{(1)} \ldots \mathbf{x}^{(p)}]^T.$$

Expanding the coordinates of X in terms of a basis in $\mathbb{R}^n$ called Φ, where Φ is any collection of vectors $[\phi_1, \phi_2, \ldots, \phi_k]$ that form a basis in $\mathbb{R}^n$, we can write any point i in X as follows:

$$\mathbf{x}^{(i)} = \sum_{k=1}^{n} \alpha_k^{(i)} \phi_k.$$

If we require our basis to be orthonormal, we can compute the coordinates of $\mathbf{x}^{(i)}$, denoted as $\alpha_k^{(i)}$ above, using the inner product:

$$\alpha_k^{(i)} = \mathbf{x}^T \phi_k = \langle \mathbf{x}, \phi_k \rangle.$$

Note that in this expansion $\langle A, B \rangle$ is simply a notation for $A^T B$ (the inner product).
In matrix form we can write the coordinates of this basis as:

$$(\mathbf{x}^{(i)})_\Phi = \Phi^T \mathbf{x}$$

so that

$$\mathbf{x}^{(i)} = \Phi \Phi^T \mathbf{x}^{(i)}.$$

For a square matrix, $\Phi\Phi^T = I$ because of the condition of orthonormality of the column vectors of Φ_i. For a rectangular matrix $\Phi\Phi^T$ is a projector into the column

[6]This section follows Hundley's work regarding the best basis [10]

space of Φ and we introduce an error if $\mathbf{x}$ is not contained in the span of the vectors ϕ_i. The error is the second term of the following equation:

$$\mathbf{x} = \sum_{j=1}^{k} \alpha_j \phi_j + \sum_{j=k+1}^{n} \alpha_j \phi_j \tag{8}$$

Thus, our error contains $n-k$ basis vectors and is defined as:

$$\mathbf{x}_{\text{err}} = \sum_{j=k+1}^{n} \alpha_j \phi_j.$$

In order to determine the best basis, we must make the following assumptions [10]:

1. The data has been mean subtracted (where the mean is in $\mathbb{R}^n$) because the best basis will have its origin at the centroid of the data.
2. The data does not fill $\mathbb{R}^n$; it lies on some linear subspace contained in $\mathbb{R}^n$.
3. The best basis should be orthonormal.

If the columns of X have been mean subtracted, we define the covariance of X, C, to be

$$C = \frac{1}{p} X^T X = \frac{1}{p} \sum_{i=1}^{p} \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^T. \tag{9}$$

Finally, we define the k-term error as the mean-squared error of the data expansion:

$$\text{Error} = \frac{1}{p} \sum_{j=1}^{p} \| \mathbf{x}_{\text{err}}^{(j)} \|^2.$$

Thus, our data can be compressed to k dimensions. We show in the rest of this section that in order to minimize the mean-squared error on the expansion $\{\phi_i\}_{j=k+1}^{k}$ we must minimize

$$\sum_{j=k+1}^{n} \langle \phi_j, C\phi_j \rangle.$$

Note that the ϕ_i form an orthonormal set, and that this is true for $k = 1$ since it is true for all k. Therefore,

$$\begin{aligned} \frac{1}{p} \sum_{j=1}^{n} \| \mathbf{x}^{(j)} \|^2 &= \sum_{j=1}^{n} \langle \phi_j, C\phi_j \rangle \\ &= \sum_{j=1}^{1} \langle \phi_j, C\phi_j \rangle + \sum_{j=2}^{n} \langle \phi_j, C\phi_j \rangle \end{aligned} \tag{10}$$

where

$$\sum_{j=2}^{n} \langle \phi_j, C\phi_j \rangle$$

is our error as per Equation 8. This error is the term we seek to minimize. Since our basis is orthonormal, we can instead find [10]

$$\max_{\phi \neq 0} \frac{\phi^T C \phi}{\phi^T \phi}.$$

We can maximize this alternate term because Equation 10 has two terms, and minimizing the second term is equivalent to maximizing the first term. Since the first term is a sum over one number, it is simply a function and is easier to maximize when written as its inner product, $\phi^T C\phi$. We divide the inner product by $\phi^T\phi$ where $\phi \neq 0$ to ensure that our maximum does not increase to infinity. To show that this is indeed the term we need to maximize, first note that if ϕ_m and ϕ_n are orthonormal,

$$\begin{aligned} \| \alpha_m\phi_m + \alpha_n\phi_n \|^2 &= (\alpha_m\phi_m + \alpha_n\phi_n)^T(\alpha_m\phi_m + \alpha_n\phi_n) \\ &= (\alpha_m\phi_m^T + \alpha_n\phi_n^T)(\alpha_m\phi_m + \alpha_n\phi_n) \\ &= \alpha_m^2\phi_m^T\phi_m + \alpha_m\alpha_n\phi_m^T\phi_n + \alpha_m\alpha_n\phi_n^T\phi_m + \alpha_n^2\phi_n^T\phi_n \\ &= \alpha_m^2 + \alpha_n^2 \end{aligned}$$

and thus

$$\| \mathbf{x}_{\text{err}} \|^2 = \sum_{j=k+1}^{n} (\alpha_j)^2. \tag{11}$$

Furthermore, if we take the jth term from our sum we have:

$$\alpha_j = {\phi_j}^T\mathbf{x}$$

and

$$\begin{aligned} \alpha_j^2 &= (\phi_j^T\mathbf{x}) \cdot (\phi_j^T\mathbf{x}) \\ &= (\phi_j^T\mathbf{x}) \cdot (\mathbf{x}^T\phi_j) \\ &= \phi_j^T(\mathbf{x}\mathbf{x}^T)\phi_j \\ &= \langle \phi_j, \mathbf{x}\mathbf{x}^T\phi_j \rangle. \end{aligned} \tag{12}$$

We rewrite the average error over p points with k basis vectors using Equations 9, 11, and 12:

$$\begin{aligned} \frac{1}{p}\sum_{i=1}^{p} \| \mathbf{x}_{\text{err}}^{(i)} \|^2 &= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{j=k+1}^{n} \| \alpha_j^{(i)}\phi_j \|^2\right) \\ &= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{j=k+1}^{n} (\alpha_j^{(i)})^2\right) \\ &= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{j=k+1}^{n} \langle \phi_j, \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T\phi_j \rangle\right) \\ &= \sum_{j=k+1}^{n}\left(\frac{1}{p}\sum_{i=1}^{p} \langle \phi_j, \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T\phi_j \rangle\right) \\ &= \sum_{j=k+1}^{n} \langle \phi_j, \left[\frac{1}{p}\sum_{i=1}^{p} \mathbf{x}^{(i)}(\mathbf{x}^{(i)T}\right]\phi_j \rangle \\ &= \sum_{j=k+1}^{n} \langle \phi_j, C\phi_j \rangle. \end{aligned}$$

This is indeed the term we minimized earlier in this discussion.

We now show that we have the best basis using eigenvalues. If the eigenvalues and eigenvectors, respectively, of the covariance matrix C are $\{\lambda_i\}_{i=1}^n$ and $\{\mathbf{v}_i\}_{i=1}^n$, where

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$$

then we rewrite C as

$$C = \lambda_1 \mathbf{v}_1 \mathbf{v}_1^T + \lambda_2 \mathbf{v}_2 \mathbf{v}_2^T + \ldots + \lambda_n \mathbf{v}_n \mathbf{v}_n^T$$

where $V^T V = V V^T = I$.
Writing our vector, ϕ, as

$$\phi = a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \ldots + a_n \mathbf{v}_n = V\mathbf{a}$$

we see that

$$\phi^T \phi = \sum_{i=1}^n (a_i \mathbf{v}_i)^T (a_i \mathbf{v}_i) = \sum_{i=1}^n a_i^2 \mathbf{v}_i^T \mathbf{v}_i = \sum_{i=1}^n a_i^2$$

and thus

$$\phi^T \phi = a_1^2 + a_2^2 + \ldots + a_n^2.$$

Furthermore,

$$\phi^T C \phi = \sum_{i=1}^n \phi^T \lambda_i \phi = \sum_{i=1}^n \lambda_i \phi^T \phi = \sum_{i=1}^n \lambda_i a_i^2$$

and thus

$$\phi^T C \phi = \lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2.$$

It follows that

$$\frac{\phi^T C \phi}{\phi^T \phi} = \frac{\lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2}{a_1^2 + a_2^2 + \ldots + a_n^2} \leq \lambda_1.$$

Rewriting

$$\lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2 \leq \lambda_1 (a_1^2 + a_2^2 + \ldots + a_n^2)$$

as

$$\lambda_1 + \lambda_2 + \ldots + \lambda_n \leq n\lambda_1$$

gives us a true inequality only when $\phi = \mathbf{v}_1$, implying that $\mathbf{v}_1$ is the best one-dimensional basis vector. The same argument applies for finding the next best basis vector, which satisfies:

$$\max_{\phi \perp \mathbf{v}_1} \frac{\phi^T C \phi}{\phi^T \phi}.$$

The fact that the second eigenvector of C is the next best vector is a consequence of Theorem 1, which says that our eigenvectors are orthogonal. To summarize the Karhunen-Loeve expansion:

Theorem 2. (The Best Basis Theorem [10]) *Suppose that:*

- *X is a $p \times n$ matrix of p points in $\mathbb{R}^n$*
- *X_m is the $p \times n$ matrix of mean-subtracted data.*
- *C is the covariance of X; $C = \frac{1}{p} X_m^T X_m$.*

Then the best (in terms of the mean-squared error defined earlier) orthonormal k-dimensional basis is given by the leading k eigenvectors of C, for any k.

It should be noted that the KL expansion is simply a method for calculating the SVD; this justifies our use of SVD to extract feature vectors. We demonstrate this by noting that the SVD of the matrix X is

$$X = U\Sigma V^T$$

where Σ is $k \times k$ and k is the rank of X. The covariance of X can be written as follows:

$$C = \frac{1}{p}X^T X = \frac{1}{p}V\Sigma U^T U\Sigma V^T = V\left(\frac{1}{p}\Sigma^2\right)V^T.$$

Thus, the best basis vectors for the rowspace of X are the right singular vectors of X, which agrees with our results from Theorem 2. Noting that

$$C = \frac{1}{p}X^T X = \frac{1}{p}U\Sigma V^T V\Sigma U^T = U\left(\frac{1}{p}\Sigma^2\right)U^T,$$

we also conclude that the best basis vectors for the rowspace of X are the left singular vectors of X. We know that the left singular vectors and right singular vectors are equivalent from our proofs regarding SVD. In other words, we know the best basis vectors of the rowspace give us the best basis vectors of the columnspace through the following relationships:

$$Xv_i = \sigma_i u_i; \qquad X^T u_i = \sigma_i v_i; \quad \lambda_i = \frac{\sigma_i^2}{p} \text{ or } \lambda_i = \frac{\sigma_i^2}{n}.$$

It is likely that our EEG data will not have zero eigenvalues because there is noise in all dimensions of our data. Without zero-valued eigenvalues, it is not straightforward to determine the appropriate rank. If there is a large gap in the values of the eigenvalues, then the rank should be chosen as i where λ_i is the eigenvalue just prior to the large drop in value. Otherwise, consider a different approach:

After considering the normalized eigenvalues

$$\lambda_i = \frac{\tilde{\lambda}_i}{\sum_{j=1}^{n}\tilde{\lambda}}$$

where $\tilde{\lambda}$ are the unscaled eigenvalues, we describe each λ_i as a percentage of the total variance captured by the i^{th} eigenspace and dimension as a function of the total variance that we want to encapsulate in a k-dimensional space. Given this, the KL dimension is the number of eigenvectors required to capture a given percentage of this total variance,

$$\text{KLD}_d = k,$$

where k is the smallest integer such that

$$\lambda_1 + \lambda_2 + \ldots + \lambda_k \geq d.$$

The value of d is chosen *ad hoc*, depending on the percentage of total variance we wish to capture [10].

4. Using the Best Basis

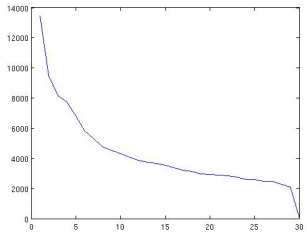

Figure 7: A plot of the values of lambda.

While it is possible to create a classifier using unmodified EEG data, the algorithm becomes complicated very quickly and is difficult and slow to compute. By manipulating our data in various ways we allow for a simpler classifier which runs more efficiently but does not contribute significantly to an increase in classification error. In the sections above, we described the SVD as a means for accomplishing this simplification. However, it is not easy to visualize what each step of the SVD does to the EEG data. Below, we manipulate the pixels of photographs to demonstrate the SVD process.

4.1. **Visualizing the SVD with Eigenfaces.**

Using the SVD, a best basis can be created with photographs of faces as a sample data set. This eigenface example will clarify and visualize the classification process. Given 30 pictures with matrices equivalent to the pixel size, 262 x 294, we concatenate each image and combine them to get one matrix of size 77028 x 30, which represents the arithmetic mean of all the images. This matrix will be the picture we recreate using SVD.

First, we find an image of the mean-subtracted faces (Figure 8) based on the picture's pixels. Then, the SVD is performed on this mean-subtracted matrix (corresponding to the mean image). A plot of the values of λ_i is shown in Figure 7. Figure 9 shows reconstructions of the original data using the first four eigenvectors. Now we can see that the first eigenvector holds most of the general features, and the subsequent eigenvectors represent more and more detailed features[7]. To create an image that is close to the original, we must determine how many λs to use. It appears that, from Figure 7, choosing between λ_{10} and λ_{20} will recover most of our data. We choose the first fifteen eigenvectors and reconstruct the original data on the first two eigenfaces after performing SVD operations. The resulting images are shown in Figure 11. The errors for each face after choosing λ_{15} as our terminal eigenvalue are also shown in Figure 11.

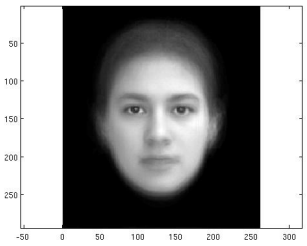

Figure 8: The mean-subtracted average eigenface (Doug Hundley, Whitman College).

Some interesting features arise from analyzing this data. First, the value of λ_1 is 0.1051, meaning the first column of the SVD matrix contains roughly 10.5% of the mean-subtracted face's data. The first 15 eigenvalues captures about 70% of

[7]If we generate the values for the first 10 lambdas we confirm the validity of this statement. They are $\lambda_1 = 0.1051, \lambda_2 = 0.0741, \lambda_3 = 0.0638, \lambda_4 = 0.0606, \lambda_5 = 0.0532, \lambda_6 = 0.0457, \lambda_7 = 0.0415, \lambda_8 = 0.0374, \lambda_9 = 0.0353,$ and $\lambda_{10} = 0.0337$.

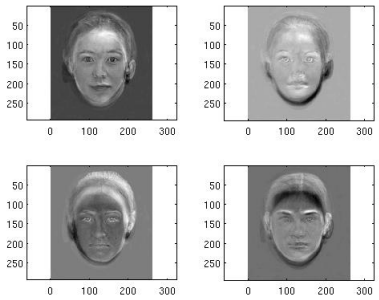

Figure 9: The first four eigenvectors of our mean-subtracted face (Doug Hundley, Whitman College).

the image's data. This is impressive considering we use only half of the columns of the SVD matrix to reconstruct the mean eigenface. Second, Figure 10 shows the coordinates (λ_1, λ_2) for the male mean eigenface SVD-reduced matrix and for the female mean eigenface SVD-reduced matrix. In this data set (but not necessarily in other sets containing images of faces) there is a distinct separation between the male faces and the females faces. It is impossible to make a conjecture about the meaning of this separation in coordinates, but it is interesting nonetheless. Finally, by viewing the images representing error in Figure 11 it is apparent which regions of the face vary widely among different people (note the eyes, lips and hairline especially). Because the SVD is only taking the best basis of vectors, the finer details are lost (though, ideally, not so many that it is noticeable).

5. Electroencephalography, Transduction, and Data Manipulation

5.1. **The EEG.** In the experiments described in this paper, electrodes are placed in standard positions as shown in Figure 12. Four frequencies of brain activity occur in humans: α rhythm from 8 to 13 hz and amplitude of 10 to 50 mV, β activity at 14 to 60 hz with amplitudes of less than 10 mV, θ waves at 4 to 7 hz and δ waves at less than 4 hz. An EEG is a reading of all of these waves; the sum of the four waves indicates whether neurons are firing in synch or irregularly, as shown

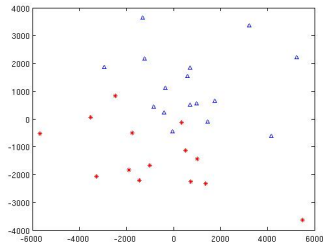

Figure 10: Coordinates (λ_1, λ_2) for the male (blue) and female (red) faces.

in Figure 15. Finally, due to the spatial resolution of the EEG and its limitations, deep brain tissue activity is not recorded.

5.2. **Organizing EEG Information in Matlab.** Within the EEG data file produced from our experiments, there exists distinct subsets of data about the five subjects, five tasks, ten trials, and seven channels of the EEG machine which stream data to the computer. The reading for each task is taken over a 2500 ms period, resulting in a 7 x 2500 matrix (7 channels, 2500 ms). We have obtained recordings from five different subjects performing five different mental tasks, each in ten separate trials. Thus, the data is arranged in a 5x5x10 cell (5 subjects, 5 tasks, 10 trials) which allows the data to be easily manipulated by subject, task and trial. For example, an image can be produced using simple Matlab code referencing any particular EEG reading. Figures 13 and 14 are images generated for task 5, trial 5, subjects 1 and 2, respectively. The five tasks are:

1 Rest: Subject is not performing any task.
2 Letter : Subject visualizes a letter being drawn.
3 Math: Subject multiplies two three-digit numbers together mentally.
4 Rotation: Subject rotates an object in his/her head.
5 Counting: Subject counts to self.

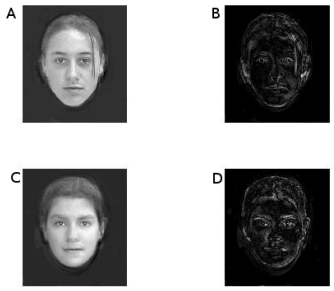

Figure 11: Results from SVD operations on the mean-valued face. A. The first eigenface reconstructed. B. Error between the first original eigenface and its reconstructed version. C. The second eigenface reconstructed. D. Error between the second original eigenface and its reconstructed version (Doug Hundley, Whitman College).

Once a classifier has been built for a specific subject, task, and trial, new sets of data can be entered to test the classifier. We should note that while it is possible to use multiple subjects or trials to build the classifier, using too much information initially can create boundaries for the classifier which are too specific to the training data, leading to a higher error when new data is presented. Another possibility is to run the SVD on the data first, hopefully eliminating some noise and improving classification error. Both methods will be explored.

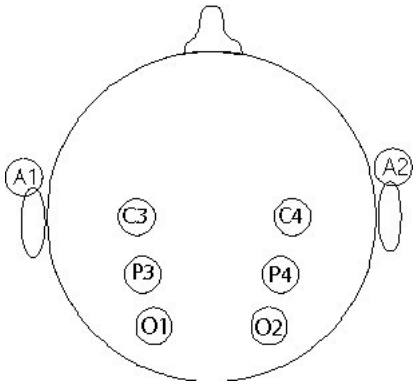

Figure 12: Positions of six electrodes on subject corresponding to six channels.

6. Using The Classifier Function in Matlab

Using Matlab, we name individual matrices which correspond to a specific subject, task and trial. Thus, 250 possible matrices can be designated. This is particularly useful when using Matlab's classify feature. The classify feature takes in three arguments : sample, training, and group. Each of these arguments indicates a specific matrix. The "sample" matrix, when we initially build our classifier, will be the same as the "training" matrix. These two matrices represent a specific subject and trial, where the rows represent data points corresponding to milliseconds from a designated time window (30 ms) and channel, and the columns represent the data

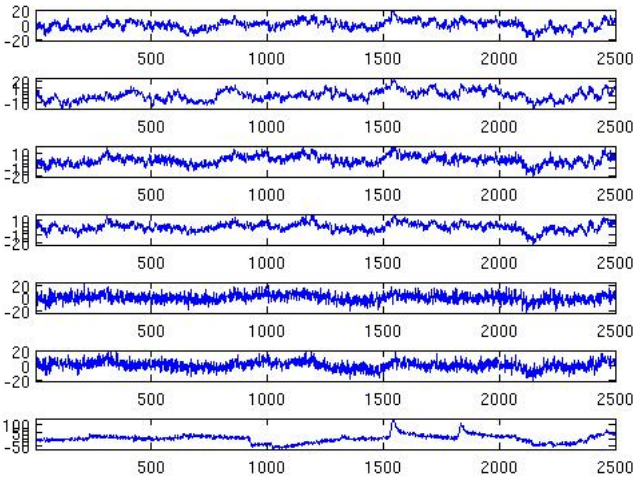

Figure 13: Subject 1, Task 5, Trial 1.

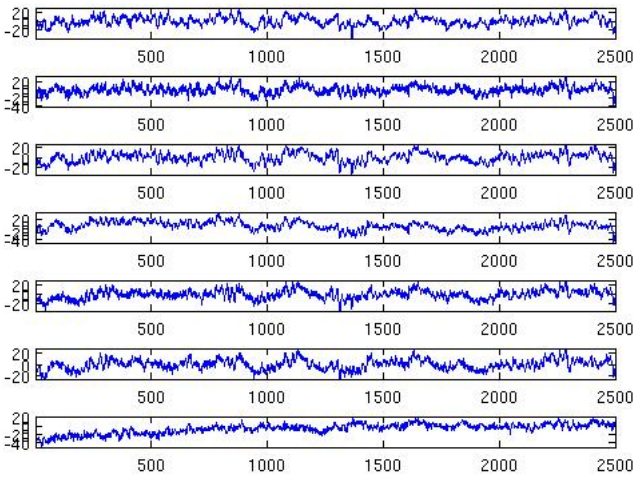

Figure 14: Subject 2, Task 5, Trial 1

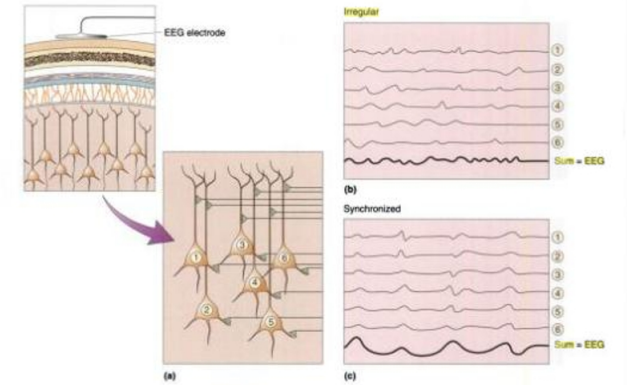

Figure 15: Bear, Mark F., Barry W. Connors and Michael A Paradiso. *Neuroscience* . 3rd ed. Lippincott, Williams & Wilkins: Baltimore, 2007.

points for all 5 tasks for a particular subject, concatenated. We lag the six channels by 3, generating 75 sample matrices for each subject of size 24×125, which returns a matrix in $\mathbb{R}^{3000}$. If SVD is performed on this "training" matrix, each window becomes 24×24 and our data points lie in $\mathbb{R}^{576}$.

The "group" matrix is a 1x2500 matrix which corresponds to the task number in each column of the training matrix. In other words, it looks like:

$$[111 \ldots 222 \ldots 333 \ldots 444 \ldots 555 \ldots]$$

where each number is repeated 75 times.

7. Visualizing the Results

After processing the EEG data in Matlab, the volume and complexity of the data points and their existence in high-dimensional space makes visualization difficult. In order to better understand and view our results, we take advantage of k-means clustering and Sammon mapping.

7.1. K-means Clustering.

To reduce the volume of data points, we utilize k-means clustering. K-means clustering uses an algorithm which requires initializing the number of patterns, n, the number of clusters, k, and a way of calculating μ_i for $i = 1 \dots k$ [4]. Ultimately, we seek to reduce the number of data points. For example, given the data points [1.1, 1.3, 0.8, 2.2, 1.7, 2.1, 3.4, 3.1, 2.9] we observe three obvious clusters at the points [1, 2, 3]. Thus, we can replace the nine data points in the original set with the three points in the cluster set. We call the points in the cluster set *cluster centers*. They are the μ_i that are found via the algorithm. This algorithm for finding the cluster centers for a set of data proceeds as outlined in Duda [4]:

1 Begin by initializing n, k, μ_i for $i = 1, 2 \dots k$
2 Sort data by proximity to the μ_i to determine which data points correspond to which cluster
3 Compute the center of mass for each cluster; the new μ_i correspond to their cluster's center of mass
4 Repeat steps 2 and 3 until there is no change in μ_i
5 Return the final μ_i for $i = 1 \dots k$

The simplicity of this algorithm will provide us with an efficient way to reduce the number of data points. We should note that there are several ways to go about step 1. One way is to randomly choose the μ_i. This can be problematic, however, because there is a possibility that empty clusters will form. To avoid this, the best initialization of the μ_i is to randomly choose k existing data points, where k is the number of clusters. Let's look at a simple example. Using the Matlab code in Appendix A.3, we can input the data points [1.1, 1.3, 0.8, 2.2, 1.7, 2.1, 3.4, 3.1, 2.9] and determine how many iterations it takes before the change in the μ_i is less than .001 (we consider that to be our "0"). The cluster centers are [3.0333 1.0000 2.1267], which we had already expected simply by observing the 9 data points earlier.

For our EEG data, there are initially 297 points in $\mathbb{R}^{576}$ for each task. By performing k-means clustering 10 times, we reduce the number of points to 50 total. Our data is initially sitting in a high-dimensional space where there are many redundancies. After k-means, we have a manageable number of data points which resemble the original data but with fewer points. This allows us to perform Sammon mapping of the data, as discussed in the following section, without overwhelming the computer.

7.2. Sammon Mapping.

Suppose that performing k-means clustering results in n data objects, each in m-dimensional (high-dimensional) space. We aim to use

Sammon mapping to find, instead, n points in 3-dimensional space[8] so that the distance between points is kept as close as possible to the original inter-point distances. That is, if $\{x_1, ..., x_n\}$ are n vectors in $\mathbb{R}^m$,we define the distance between x_i andx_j as $d_{ij} = \|x_i - x_j\|$. If we let $\{y_1, y_2, ..., y_n\}$ be the 3-dimensional representations (respectively) of the x_i, then we define $\delta_{ij} = \|y_i - y_j\|$. The goal of the Sammon map is to find the coordinates $\{y_1, ..., y_n\}$ in $\mathbb{R}^2$ so that $d_{ij} = \delta_{ij}$.

Of course, we do not expect that we will be able to get equality; therefore, we want to find $\{y_1, ..., y_n\}$ so that the following "stress function" E is minimized:

$$\mathrm{E} = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\delta_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{(\delta_{ij} - \mathrm{d}_{ij})^2}{\delta_{ij}}.$$

This error function calculates the difference in configurations between the n points in m-dimensional space and between the n points in 3-dimensional space (badness-of-fit), returning a value in $[0, 1]$ where 0 indicates lossless mapping [9]. We notice that E is a real valued, non-negative function that depends on $3n$ variables (n data points, y_k, in $\mathbb{R}^3$). We can numerically find the minimum by using gradient descent (move in the direction opposite the gradient). From k-means clustering, we know we have $n = 50$ data points, leaving us to solve 150 variables. If we had not performed Sammon mapping, we would have had to solve $3n = 3(297 \cdot 5) = 4455$ variables (which would overwhelm the computer). After Sammon mapping our data now lies in $\mathbb{R}^3$ instead of $\mathbb{R}^{576}$.

8. Testing the Classifier

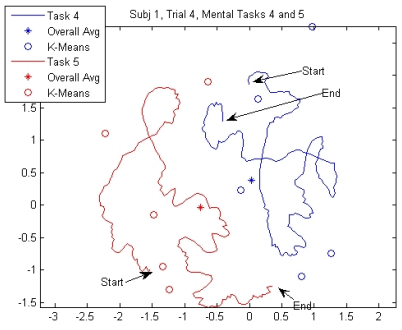

Figure 16: Tasks four and five for subject one.

Before running statistics on the classification of our EEG training data, we must first ensure that the classifier is working properly. Some questions to answer include whether or not the preprocessing is effective, if enough noise has been removed to distinguish between tasks, and how well the processor is splitting the data. One way to do this is through k-means clustering. In Figure 16, we compare tasks four and five for subject one, trial four. The open circles demonstrate the k-means cluster centers, with the stars designating the overall class mean for each task. Finally, the lines designate the moving average of 50 data points in the subject's brain mapping over a ten second period. Sammon mapping projects the results into 2-dimensional space. Thus, Figure 16 demonstrates that our data is indeed being classified into distinct tasks with little to no overlap. However, an live-streaming version of the classification procedure may result in some difficulty, since the classes do not look as nice if we take only a portion of the curves. Thus,

[8]Sammon mapping can be used to find n points in d-dimensional space with $d < m$, but for visualization we choose $d = 3$.

this image confirms that we should take into account the long-term time dependence of the signal. Another potential problem to address is the possibility that initializing k-means clustering differently will produce different results. To confirm that this does not happen, we use Sammon mapping. Figure 19 displays the results of k-means clustering performed 10 times on each task (subject one, trial four) in $\mathbb{R}^3$ and projected into two-dimensional space via Sammon mapping. We can see that while the results differ slightly when the initial cluster centers are randomly assigned, there is still a clear distinction between classes. In Figure 17, we compare two different subjects (subjects 1 and 3) performing task four. This determines whether or not the classifier must be trained for each subject. Indeed, the figure shows distinct separation between the two subjects, confirming a need to retrain the classifier each time a new subject is introduced. We have now confirmed that it is possible to classify our EEG data using the process described in this paper with reasonable accuracy.

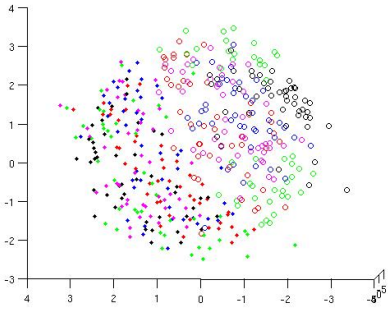

Figure 17: Subjects 1 and 3 performing task four, where k-means clustering was performed multiple times and mapped via Sammon mapping.

8.1. **Using Training Data to Determine the Effectiveness of the Classifier.** When choosing vectors of data to classify, we can either use real-time linear selection or randomly permute the vectors first. Because the first option did not produce desirable classification rates, we randomly permute the vectors instead. The classification process is coded in Matlab to work as follows: First, we pre-process our EEG data. This not only performs the SVD, but introduces a lag and a shift. Figure 18 demonstrates that a larger lag produces better classification rates, so we chose a lag of 15 for our final test[9]. Secondly, we choose how much data to use as training data and how much data to use to test the classifier. Ultimately, we use 80% to train and 20% to classify, but Table 1 shows us that it is somewhat arbitrary how we split the data until we use very little data (10%) to train the classifier (CR=classification rate).

.

Finally, we run Matlab's classifier function on the data, using a quadratic boundary (as opposed to linear, which had poor classification rates). If the classification is run 10 times and the rates are averaged, we find that the five tasks are classified with the rates show in Table 2.

[9]To prove that this correlation between higher lag and better classification rate was not by chance, we ran the test across all tasks for subject 3.

Table 1: Classification Rates (CR) for various ratios of training data vs. classification data.

% Training	95	60	30	10
% Task 1 CR	93.31	91.63	85.62	52.86
% Task 2 CR	90.54	89.24	84.27	58.03
% Task 3 CR	99.02	98.61	98.68	95.41
% Task 4 CR	99.65	99.71	99.53	93.90
% Task 5 CR	99.45	99.67	99.58	99.12

Table 2: Classification rates averaged over 10 runs using 80% of the data for training.

Trial	1	2	3	4	5
% Correct	0.9336	0.8980	0.9802	0.9969	0.9950

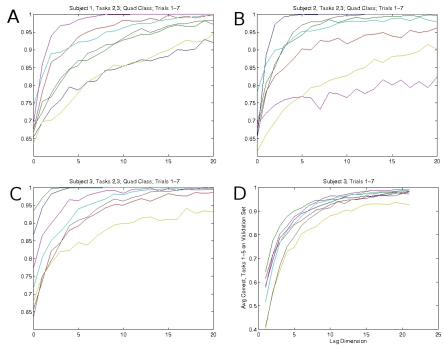

Figure 18: As the lag increases, the classification rate improves. Figure D demonstrates that the correlation is not by chance by running the test on all tasks for subject 3.

Finally, the following confusion matrix demonstrates how often a task, i (column i = task i), is confused with another task, j (row j = task j):

$Task(\%)$	1	2	3	4	5
1	96.5	4.5	2.2	1.8	0.0
2	1.7	94.5	3.0	1.8	2.0
3	1.7	0.9	94.8	0	0
4	0.0	0.0	0	96.4	1.3
5	0.0	0.0	0	0	96.7

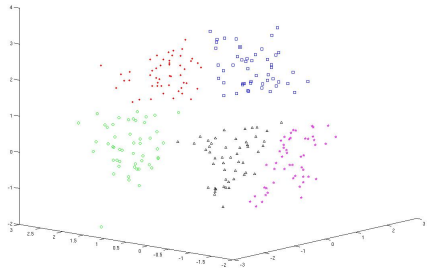

Figure 19: Results of k-means clustering performed 10 times on each task in trial four, subject one. Task 1=red dots. Task 2=green circles. Task 3=blue squares. Task 4=purple stars. Task 5=black triangles.

Conclusion

In a recently published paper, Sheena Luu and Tom Chau [12] explore the use of Near-infrared spectroscopy (NIRS) as an alternative to the EEG cap for the same purpose of applying functional applications to a subject's thoughts. Their experiment was of a simple design–subjects were asked which of two drinks they preferred–but the results were promising: their classification process was able to determine a subject's preference accurately 80% of the time. NIRS is non-invasive and requires less set-up than the EEG, and it can read signals quickly, accurately, and effectively. Essentially, NIRS measures changes in the color of the brain as oxygen is delivered via hemoglobin to different regions. As a region becomes active, blood flow to the area increases, resulting in increased hemoglobin[13] (the iron in hemoglobin affects the color) . The portability and simplicity of this system (and better spatial resolution) may offer a better tool for this line of research in future experiments.

Regardless of the tool used to measure brain signaling, building a classifier is a complex process. In order to avoid implementing a complicated classifier, recall that we split our data into windows of time and performed SVD, taking only the best basis of information and eliminating noise. This greatly simplified the work to be done in Matlab. To visualize our results, we employed k-means clustering and Sammon mapping. Through those two methods, it is clear that our pre-processing techniques are effective enough to allow for reasonable classification rates, that each task is recognized as separate from the others, and that randomly assigning initial cluster centers does not effect the outcome of the classification. The visualization of our data also confirmed the need to initialize the classifier on each new subject. Statistical analysis also showed that our classifier is also effective. Overall, using 80% of our data as training data resulted in a classification rate of 96.06% for the remaining 20%, averaged over all of the tasks. The tables in Section 8.2 confirm that our classification rates are not only satisfactory, but exceptional.

Our classifier is now ready for new data. In the future, we hope to explore the classifier's accuracy on new subjects and use newly collected EEG information to

fine-tune the classifier. There will be one problem that we will have to overcome, which is the discrepancy between streaming live EEG data and randomly permuting the processed EEG vectors. Obviously, we cannot permute the vectors until the data has been collected, making real-time processing impossible. If we want to classify in real time, creating a solution to this problem will be necessary before further progress can be made. Because our data is non-stationary (the statistics for the data change over time), this is not an easy task. It requires creating algorithms which can track the changing subspaces.

References

[1] "Comparison of linear, nonlinear, and feature selection methods for EEG signal classification", D. Garrett, D.A. Peterson, C.W. Anderson, and M.H. Thaut. in *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 11:12, 2007.

[2] "Connecting Your Brain to the Game: Using an EEG cap, a startup hopes to change the way people interact with video games", K.Greene. In *Technology Review*, MIT press, `http://www.technologyreview.com/business/18276/page1/`, 2007.

[3] "Classification of Time-Embedded EEG Using Short-Time Principal Component Analysis", C. Anderson, M. Kirby, D. Hundley and J. Knight. In *Towards Brain-Computer Interfacing*, edited by G. Dornhege, J. del R. Millan, T. Hinterberger, D.J. McFarland and K.-R. Muller, MIT Press, p 261-278, 2007.

[4] Duda, Richard O, Peter E. Hart and David G. Storke. *Pattern Classification*. 2nd ed. John Wiley & Sons, Inc: New York, 2001.

[5] Fukunaga, Keinosuke. *Introduction to Statistical Pattern Recognition*. 2nd ed. Academic Press: San Diego, 1990.

[6] Weisstein, Eric W. "Central Limit Theorem." From *MathWorld–A Wolfram Web Resource*. `http://mathworld.wolfram.com/CentralLimitTheorem.html`.

[7] "Probability Mass Function". (2009). In *EncyclopÃędia Britannica*. Retrieved February 11, 2009, from EncyclopÃędia Britannica Online: `http://www.britannica.com/EBchecked/topic/477521/probability-mass-function`

[8] Lay, David C. *Linear Algebra and Its Applications*. 3rd ed. Addison Wesley: Boston, 2003.

[9] De Ridder, Dick. "Sammon Mapping". 1999: `http://www.ph.tn.tudelft.nl/Research/neural/feature_extraction/papers/asci99b/node2.html`

[10] Hundley, Doug R. "Chapter Six: The Best Basis". *Introduction to Emperical Modeling*. `http://people.whitman.edu/~hundledr/courses/M350/Chap6A.pdf`

[11] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297

[12] Luu, Sheena and Tom Chau. "Decoding Subjective Preference From Single Trial, Near-infrared Spectroscopy Signals". *Journal of Neural Engineering* 6: 2009, 16003-16011.

[13] Banaji M, Mallet A, Elwell CE, Nicholls P, Cooper CE (2008) "A Model of Brain Circulation and Metabolism: NIRS Signal Changes during Physiological Challenges". PLoS Comput Biol 4(11): e1000212. doi:10.1371/journal.pcbi.1000212

Appendix A. MATLAB CODE

A.1. **1. Code from Section 1.2.** The following code solves the matrix equation in Section 1.2:

```
% In the array X, the first row are the x-coordinates, the second row are
% the y-coordinates of the data

X=[1 1 2  2 -1 -2 -1 -2
   1 2 -1 1  2  1 -1 -2]

%The vector T gives the desired target values (Class 1=1, Class 2=-1)
T=[1 1 -1 -1 1 1 -1 -1]

%Solve Ax=b means:
[m,n]=size(X)  %Matrix A will be n x 3:
A=[X' ones(n,1)]
x=A\T'   %x contains three numbers- a, b and c.

a=x(1); b=x(2); c=x(3);
xx=linspace(-3,3,150);  %Domain values to plot the line.
if b~=0
    y=(-c-a*xx)/b;
end
```

The following output is generated:

```
X =

     1     1     2     2    -1    -2    -1    -2
     1     2    -1     1     2     1    -1    -2

T =

     1     1    -1    -1     1     1    -1    -1

m =

     2

n =

     8

A =

     1     1     1
     1     2     1
     2    -1     1
     2     1     1
    -1     2     1
```

```
    -2     1     1
    -1    -1     1
    -2    -2     1

x =

   -0.2247
    0.6235
   -0.2338
```

where $x = [a\ b\ c]^T$.

A.2. **2. Code from Section 2.2.** After defining data in two classes, Class1Sample and Class2Sample, as well as the matrices corresponding to μ_i and σ_i, the following code generates a matrices which indicates to which class each $x|x \in$ Class1Sample, Class2Sample has been assigned.

```
X=[Class1Sample; Class2Sample];  %1000 x 1
G1=-(X-mu(1)).^2/(2*(sigma(1)^2))-log(sigma(1));
G2=-(X-mu(2)).^2/(2*(sigma(2)^2))-log(sigma(2));
G=[G1 G2];  %1000 x 2
[tmp class]=max(G,[],2);
```

The following code calculates the minimum distance between all x and μ_i where $i = 1, 2$:

```
Y1=abs(X-mu(1));
Y2=abs(X-mu(2));
[tmp Z]=min([Y1,Y2],[],2);
```

If we want to view the results and percent error of these two classifications we use the following code:

```
figure(2)
plot(class,'b*'); hold on;
plot(Z,'ro'); hold off

Targets=[ones(500,1); 2*ones(500,1)];
Percent1=round(100*(sum(abs(Targets-class)))/1000)
Percent2=round(100*(sum(abs(Targets-Z)))/1000)
```

A.3. **Code from Section 7.1.** Below is the code to implement k-means clustering.

```
%Example:  k-means clustering
%Original data, dimension x number of points
x=[1,1.01,0.99,2.3,2.1,1.98,3.1,3.01,2.99];
n=length(x);
k=3;

%c=randn(1,3); %Initialize the cluster centers

tt=randperm(n);
c=[x(tt(1)), x(tt(2)), x(tt(3))];
oldc=100*ones(1,3);  %Some big numbers
```

```
Distances=mean(sum((c-oldc).^2,1))
while Distances>0.001
    %Sort the j^th data point.  We'll use an index vector
    for jj=1:n
      for kk=1:3
        dd(kk)=norm(x(:,jj)-c(:,kk));
      end
      if dd(1)==min(dd)
          Temp(jj)=1;
      elseif dd(2)==min(dd)
          Temp(jj)=2;
      else
          Temp(jj)=3;
      end
    end

    oldc=c;
    idx1=find(Temp==1);
    c(:,1)=sum(x(:,idx1))/length(idx1);
    idx2=find(Temp==2);
    c(:,2)=sum(x(:,idx2))/length(idx2);
    idx3=find(Temp==3);
    c(:,3)=sum(x(:,idx3))/length(idx3);
    Distances=mean(sum((c-oldc).^2,1))
end
```

Appendix B. ALTERNATE PROOF TO THE KL THEOREM

We can prove through an alternate method that we have the best basis. Given any orthonormal basis, $\{\psi_i\}_{i=1}^{p}$ we know that:

$$\frac{1}{p}\sum_{i=1}^{p} \parallel \mathbf{x}^i \parallel = \sum_{j=1}^{D} \psi_j^T C \psi_j + \sum_{j=D+1}^{p} \psi_j^T C \psi_j$$

where C is the covariance matrix. We also know that this sum is constant. Because of this, minimizing the second term of the sum is equivalent to maximizing the first term of the sum. Using the eigenvector basis we can say:

$$\sum_{j=1}^{D} \phi_j^T C \phi_j = \lambda_1 + \ldots + \lambda_D,$$

and we can evaluate the second term as

$$\frac{1}{p}\sum_{i=1}^{p} \parallel \mathbf{x}_{\text{err}}^i \parallel^2 = \sum_{j=D+1}^{p} \phi_j^T C \phi_j = \lambda_{D+1} + \ldots + \lambda_p.$$

Furthermore, we can show that the first D eigenvectors of C form the best basis over all D-term expansions, as we stated earlier.

Let $\{\psi_i\}_{i=1}^{p}$ be some other D-dimensional basis. Each ψ_i can be written in terms of its coordinates with respect to the eigenvectors of C,

$$\psi_i = \sum_{k=1}^{D} (\psi_i^T \phi_k)\phi_k = \Phi\alpha_i$$

so that $\alpha_{ik} = \phi_i^T \phi_k$. It follows that:

$$\psi_i^T C \psi_i = \alpha_i^T \Lambda \alpha_i$$

Let us now consider the mean squared projection of the data onto this D-dimensional basis:

$$\sum_{j=1}^{D} \psi_j^T C \psi_j = \sum_{j=1}^{D} \alpha_j^T \Lambda \alpha_j = \sum_{j=1}^{D} \lambda_1 \alpha_{j1}^2 + \ldots + \lambda_p \alpha_{jp}^2$$

so that

$$\sum_{j=1}^{D} \psi_j^T C \psi_j = \lambda_1 \sum_{j=1}^{D} \alpha_{j1}^2 + \lambda_2 \sum_{j=1}^{D} \alpha_{j2}^2 + \ldots + \lambda_p \sum_{j=1}^{D} \alpha_{jp}^2.$$

Now, consider the coefficients

$$\{\alpha_{1i}, \ldots, \alpha_{Di}\} = \{\psi_1^T \phi_i, \ldots, \psi_D^T \phi_i\}$$

as the coefficients from the projection of ϕ_i onto the subspace spanned by the ψ's:

$$\mathrm{Proj}_{\psi}(\phi_i) = \alpha_{1i}\psi_1 + \ldots + \alpha_{Di}\psi_D.$$

We can show that

$$\sum_{j=1}^{D} \alpha_{ji}^2 = \| \mathrm{Proj}_{\psi}(\phi_i) \|^2 \leq 1$$

is true if and only if ϕ_i is in the span of the columns of ψ. The maximum of

$$\lambda_1 \sum_{j=1}^{D} \alpha_{j1}^2 + \lambda_2 \sum_{j=1}^{D} \alpha_{j2}^2 + \ldots + \lambda_p \sum_{j=1}^{D} \alpha_{jp}^2$$

is found by replacing the coefficients of the first D terms with 1 and the remaining with zero. This corresponds to the error found by using the eigenvector basis.