ABSTRACT

A COMPARATIVE STUDY FOR THE HANDWRITTEN DIGIT RECOGNITION PROBLEM

By

José Israel Pacheco

May 2011

The problem of handwritten digit recognition has long been an open problem in the field of pattern classification and of great importance in industry. The heart of the problem lies within the ability to design an efficient algorithm that can recognize digits written and submitted by users via a tablet, scanner, and other digital devices. In this thesis project, we will compare the success rate of three algorithms on the publicly available and widely used MNIST database. In particular, a SVD-based algorithm and a linear approximation model will be reviewed and implemented, while a Grassmann framework will be introduced and tested against the aforementioned techniques. We anticipate that the Grassmann framework will achieve a success rate higher than that of the previously established algorithms mentioned above.

A COMPARATIVE STUDY FOR THE HANDWRITTEN DIGIT RECOGNITION PROBLEM

A THESIS

Presented to the Department of Mathematics and Statistics

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Applied Mathematics

Committee Members:

Jen-Mei Chang, Ph.D
Tangan Gao, Ph.D
Hojin Moon, Ph.D

College Designee:

Robert Mena, Ph.D.

By José Israel Pacheco

B.S. Mathematics, 2007, Chapman University

May 2011

WE, THE UNDERSIGNED MEMBERS OF THE COMMITTEE,

HAVE APPROVED THIS THESIS

A COMPARATIVE STUDY FOR THE HANDWRITTEN DIGIT RECOGNITION PROBLEM

By

José Israel Pacheco

COMMITTEE MEMBERS

Jen-Mei Chang, Ph.D	Mathematics & Statistics
Tangan Gao, Ph.D	Mathematics & Statistics
Hojin Moon, Ph.D	Mathematics & Statistics

ACCEPTED AND APPROVED ON BEHALF OF THE UNIVERSITY

Robert Mena, Ph.D.
Department Chair, Department of Mathematics and Statistics

California State University, Long Beach

May 2011

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Jen-Mei Chang for her guidance and patience during my tenure at California State University, Long Beach which has allowed me to achieve success both in and out of the classroom. Under her tutelage, I have been able to develop both as a student and as a researcher.

I would also like to thank Cecile Lindsay, Vice Provost of Academic Affairs and Dean of Graduate Studies at CSU, Long Beach, for supporting my research via the Graduate Research Fellowship.

Lastly, to my mother and grandparents: *Gracias por todo el amor y el apoyo que me han dado durante mi carera academica. Ningunos de mis exitos serian posibles sin ustedes. Mi maestria es mas de ustedes que mia. Le doy gracias a Dios por tenerlos a mi lado.*

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER 1

INTRODUCTION

A classic problem in the field of pattern recognition is that of handwritten digit recognition. Suppose that we have an image of a digit submitted by a user via a scanner, a tablet, or other digital device. The goal is to design an algorithm that can correctly identify the digit. The applications of such an algorithm are far reaching. With this technology, the post office would be able to scan envelopes and effectively sort them by zip code and banks would be able to process checks more efficiently [1]. Handwritten digit recognition is really a subproblem of handwritten character recognition where an algorithm is needed not only to classify digits, but letters as well. Findings in the field of digit recognition can surely help advance that of characters with perhaps one of the most interesting applications being the ability to convert a document written by a user on a tablet into a more readable document. There are essentially two types of algorithms which one can design: memory based and learning based.

Suppose we have a training set of digits, that is, a set of images whose classification as a certain digit is known. Memory based algorithms store theses images and classify an unknown digit by comparing it to each of the stored patterns. Instead of storing the training set, learning based algorithms try to learn from the known patterns and build a classification function accordingly. An example of a learning based algorithm is a neural network which has been proven to be very successful in the handwritten digit recognition [1]. In this thesis, we will only discuss memory based algorithms. While it has not been shown that these

algorithms alone can be as successful as neural networks in classifying digits, they do offer great insight into the inherent geometry of the data and thus, provide a platform for the application of several mathematical concepts as we will see in this discussion.

Every algorithm that will be presented in this thesis is, in some form or another, a *nearest neighbor algorithm*. Suppose we have ten patterns in our training set, $x_1, x_2, \ldots, x_{10}$, an unknown pattern y, and a metric $d(x, y)$. We calculate $d(x_i, y)$ for $i = 1, \ldots, 10$ and classify it as a pattern of type x_k where k is the index that minimized $d(x_i, y)$. In other words, y is classified as the pattern that it is closest to, hence the phrase nearest neighbor.

In Chapter 1, we present a simple yet effective algorithm which assumes that each set of digits lies in subspace whose basis is obtained via the idea of *Singular Value Decomposition* (SVD). When an unknown digit is read in, we project the digit onto each of the ten subspaces and classify the digit according to the smallest residual vector under the 2-norm. In Chapter 2, we assume a more general model for our data. Here we treat each digit as a point on a high dimensional manifold and and use the tangent plane at that point as an invariant feature for comparison. The metric used to determine the nearest neighbor is *tangent distance* which is defined as the Euclidean distance between two tangent planes. In Chapter 3, we present the concept of *principal angles* and how they are used to obtain metrics on the Grassmann manifold. When then introduce an algorithm that models each digit as a vector in a subspace which in turn is a point on the Grassmann manifold. We then experiment with different metrics on the Grassmannian to see which yields the best results. Lastly, in Chapter 4, we test the algorithms discussed on the MNIST database [2] and present the results. MNIST is a database handwritten digits collected by Yann LeCun of the Courant

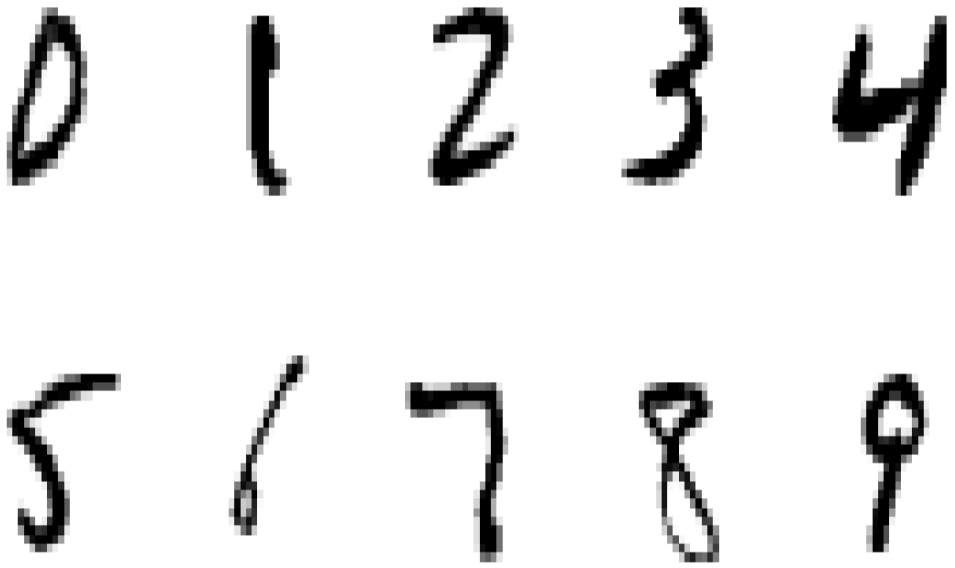
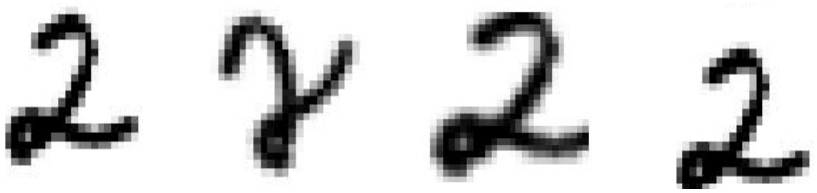


FIGURE 1. Samples of digits from MNIST's training set.

Institute at New York University and Corinna Cortes of Google Labs, New York. The database consists of 60,000 training digits and 10,000 testing digits, all of size 28×28 pixels. The digits have all been size-normalized and centered. Figure 1 shows a few samples from MNIST's training set.

Throughout this study we will utilize three standard transformations of images: rotations, scaling, and translations. In Figure 2(a), we have an image of the digit two taken from MNIST's database. Figure 2(b) has the same digit, but it has been rotated counterclockwise by $\frac{\pi}{4}$ while the digit in (c) has been scaled by a factor of 1.5 and translated 5 pixels down in (d). MATLAB implementations of these transformations can be seen in Codes #4, #5, and #6 of the Appendix.



(a) Original (b) Rotated (c) Scaled (d) Translated

FIGURE 2. Rotation, scaling, and translations of an image.

CHAPTER 2

CLASSIFICATION WITH SINGULAR VALUE DECOMPOSITION

Theory

The Singular Value Decomposition is a standard technique used in data analysis for the purpose of dimensionality reduction. Here it will be used as a tool for classification. Before we delve into the details of its application, let us first review some of the theoretical background about singular value decomposition (SVD).

Theorem 2.1. *(Singular Value Decomposition) Let A be a real $m \times n$ matrix and $d = min\{m,n\}$. Then there exist orthogonal matrices U and V such that*

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

where $U \in \mathbb{R}^{m\times m}$,$V \in \mathbb{R}^{n\times n}$, and $\Sigma = diag(\sigma_1, \ldots, \sigma_d) \in \mathbb{R}^{m\times n}$.

The σ_i's are known as the *singular values* of A and the columns of U and V are called the *left and right singular vectors* of A, respectively. As we see in Theorem 2.2, these vectors are of great importance.

Theorem 2.2. *(Fundamental Subspaces) Let A be a real $m \times n$ matrix with rank r. Let $\mathcal{R}(A)$ and $\mathcal{N}(A)$, denote the range and null space of A, respectively. Then,*

1. *The left singular vectors $u_1, ..., u_r$ of A are an orthonormal basis in $\mathcal{R}(A)$ and $rank(A) = dim(\mathcal{R}(A)) = r$.*

2. *The right singular vectors $v_{r+1}, ..., v_n$ of A are an orthonormal basis in $\mathcal{N}(A)$ and $dim(\mathcal{N}(A)) = n - r$.*

3. *The right singular vectors* $v_1, ..., v_r$ *of* A *are an orthonormal basis in* $\mathcal{R}(A^T)$.

4. *The left singular vectors* $u_{r+1}, ..., u_m$ *of* A *are an orthonormal basis in* $\mathcal{N}(A^T)$.

The first result of Theorem 2.2 says that taking the SVD of a matrix produces a basis for its column space. This fact is at the heart of our SVD-based algorithm. Another useful property of SVD is its approximation property described in Theorem 2.3.

Theorem 2.3. *Let* A *be a real* $m \times n$ *matrix with rank* r. *The matrix approximation problem*

$$\min_{\text{rank}(Z)=k} \|A - Z\|_2$$

has the solution

$$Z = U_k \Sigma_k V_k^T$$

where $k < r$, $U_k = [u_1|\dots|u_k]$ *(the first* k *left singular vectors of* A*),* $V_k = [v_1|\dots|v_k]$ *(the first* k *right singular vectors of* A*),* $\Sigma_k = diag(\sigma_1, \dots, \sigma_k)$ *(the first* k *singular values of* A*).*

In words, the best rank-k approximation of a matrix in the 2-norm sense is provided by SVD. SVD also solves the minimization problem under the Frobenius matrix norm. Proofs of these results can be found in [3].

Implementation

Let us now introduce a convenient way of representing an image in order to use these results. A gray-scale image, A, of resolution $m \times n$ is essentially an $m \times n$ matrix whose entries correspond to the gray level of the corresponding pixel. On an 8-bit machine, the possible gray level values are 0 to 255 with 0 corresponding to black and 255 corresponding to white. For an illustration see Figure 3.

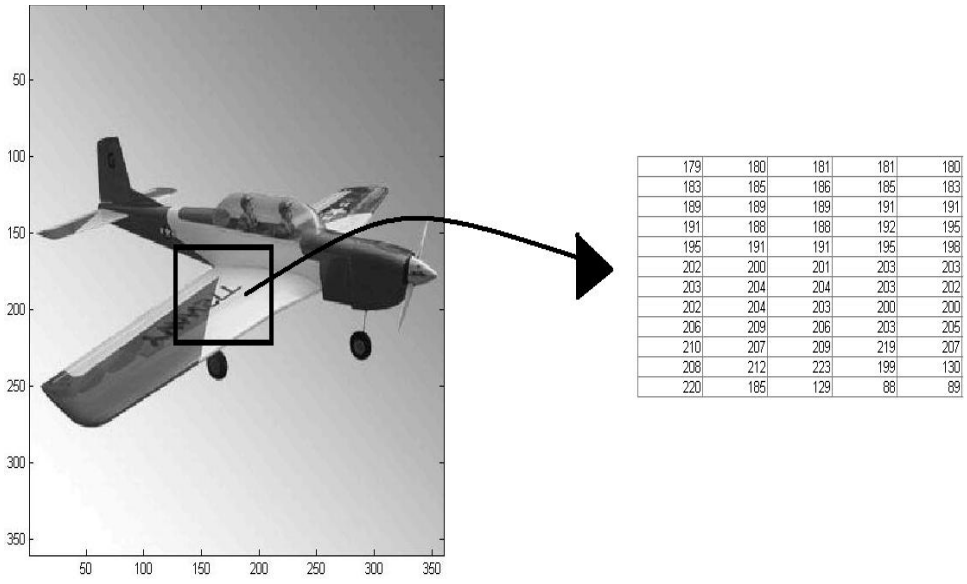


FIGURE 3. Image represented as a matrix.

Let $a_1, \ldots, a_n$ denote the columns of our matrix A. Since A is an $m \times n$ matrix, each a_i is a vector in $\mathbb{R}^m$. If we concatenate the columns of A, we can represent the matrix as a single vector in $\mathbb{R}^{m \cdot n}$.

$$A = [a_1 | \ldots | a_n] \longrightarrow \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

With this representation, we can easily store multiple images in a single data matrix by placing an image in every column as is illustrated in Figure 4 where $x_1, x_2, \ldots, x_n$ are images that have been concatenated.

Keeping these representations in mind, suppose we have p images of the number 2 with resolutions $l = m \times n$. Concatenating these images will yield p vectors in $\mathbb{R}^l$. Let us now store these vectors in the columns of a data matrix, X, as was done in Figure 4. These columns now span a linear subspace of $\mathbb{R}^l$ (i.e. the column space of X). By Theorem 2.2, the left singular vectors of X form an

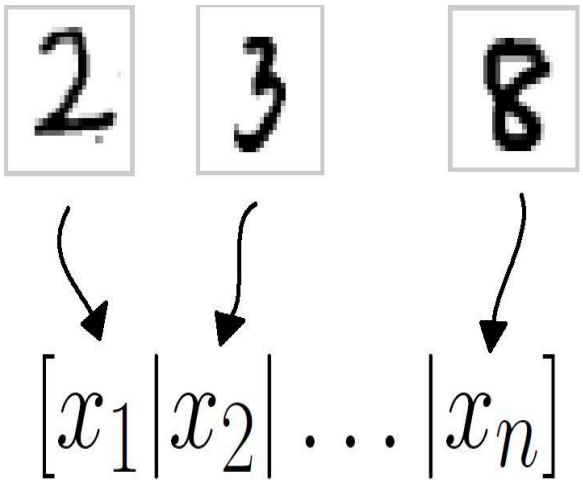


FIGURE 4. Data matrix whose columns are images that have been concatenated.

orthonormal basis for this space. Moreover, from Theorem 2.1 we know that we have a total of l left singular vectors. However, we will be able to sufficiently represent this subspace using only a few of the left singular vectors. By few we mean much smaller than l. To see why this is true, let us look at an example.

Suppose we take 50 images of the digit two from the MNIST database and arrange them in a data matrix as described above. Since the images have resolution 28×28, this yields a 784×50 data matrix. Taking the SVD of the matrix and examining the singular values, we see that there is a significant drop in magnitude after the first singular value as is seen in Figure 5.

The singular values represent the distribution of the data's energy among the singular vectors. Since the first singular value is drastically larger than the other singular values we would expect the first left singular vector to exhibit the dominant behavior of the data, that is, we would expect the vector to look like a number two after we reshaped it back into a 28×28 image. If we look at Figure 6(a), this is indeed what we obtain. Figure 6 (b) and (c) are the second and third singular vectors which represent variations in the data.

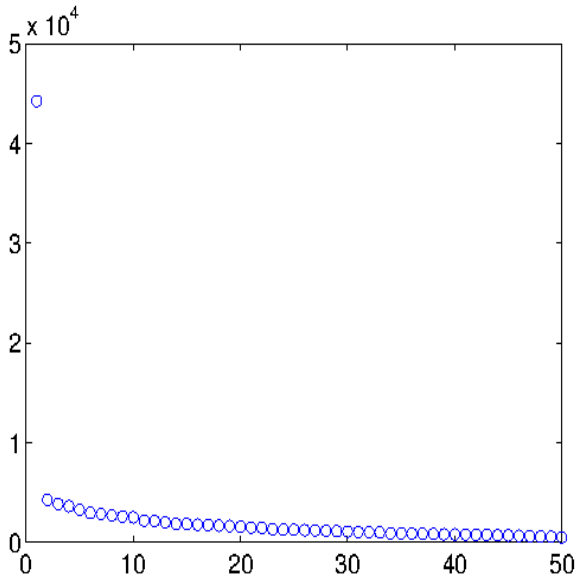
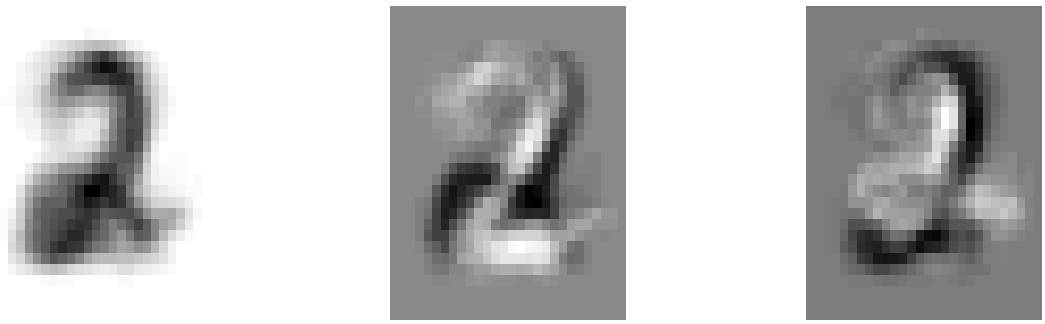


FIGURE 5. Singular value distribution of data matrix with a '2' in each column.



(a) First Singular Vector (b) Second Singular Vector (c) Third Singular Vector

FIGURE 6. First three singular vectors of data matrix with a '2' in each column.

Since each left singular vector has a singular value associated with it each vector has an energy attributed to it. Consider the following definition:

Definition 2.4. *(Cumulative Energy) Let A be a matrix of rank r with singular values $\sigma_1 \dots \sigma_r$. The cumulative energy of the first t, $(1 \leq t \leq r)$, singular values is*

$$\frac{\sum_{i=1}^{t} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}. \tag{2.1}$$

In this example, the first ten left singular vectors of the data matrix obtain a cumulative energy greater than 97% which is a common threshold for determining numerical rank [4]. Hence, even though the data lies in the high dimensional space of $\mathbb{R}^{784}$, a majority of the information can be represented using a few basis vectors. In general, suppose that the first k left singular vectors of our data matrix, $u_1, \dots, u_k$, capture 97% of the cumulative energy. These vectors will form the basis for the "digit space", that is, the subspace of all images of the same digit. If we have an unknown digit P, we would like to calculate its distance from the digit space which is defined to be

$$\min_{\alpha} \|U_k \alpha - P\|_2 \tag{2.2}$$

Where $\alpha \in \mathbb{R}^k$ and U_k is the matrix whose columns are $u_1, \dots, u_k$. Instead of solving this minimization problem, we can equivalently solve for the square of the 2-norm.

$$\begin{aligned}
\min_{\alpha} \|U_k\alpha - P\|_2^2 &= \min_{\alpha}(U_k\alpha - P)^t \cdot (U_k\alpha - P) \\
&= \min_{\alpha}(\alpha^t U_k^t - P^t) \cdot (U_k\alpha - P) \\
&= \min_{\alpha}(\alpha^t\alpha - \alpha^t U_k^t P - P^t U_k\alpha + P^t P)
\end{aligned}$$

Taking the derivative of the last expression with respect to α and setting it equal to zero we get

$$\begin{aligned}
2 \cdot \alpha^t - 2 \cdot P^t U_k &= 0 \\
\alpha^t &= P^t U_k \\
\alpha &= U_k^t P
\end{aligned}$$

Intuitively, $U_k^t P$ is the projection of P onto the digit space so the distance is just the 2-norm of the residual vector. In Figure 7, we have a geometric illustration of the scenario where $S = \text{span}(U_k)$.

We can now use this as a means of classifying unknown digits. We can calculate the distance from an unknown digit to all ten digit spaces (one for each digit) and classify the probe based on the smallest residual. For instance, if the distance from P to two space is smaller than the distance to any other digit space, then P will be classified as a two. This algorithm is described in full in Algorithm 1.

In order to determine k, the number of singular vectors to be used, we take the SVD of each of the ten data matrices and find the number of vectors required

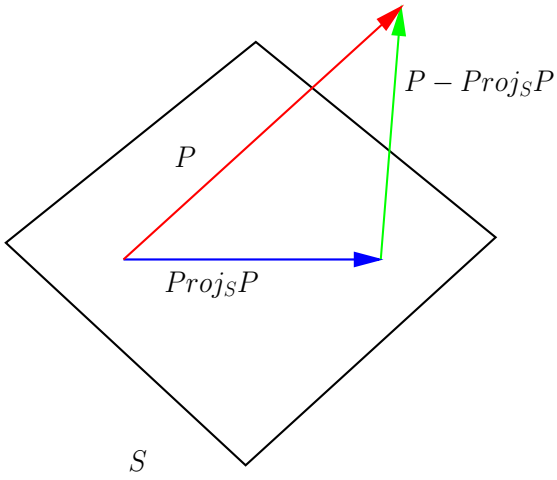


FIGURE 7. The probe P, its projection onto $S = \text{span}(U_k)$, and the residual.

Algorithm 1 Digit Classification with SVD [3]
Input: Probe image P
Output: Classification of P as a digit $0, \ldots, 9$.
Step 1: Construct the data matrix for each digit using the training images.
Step 2: Calculate the SVD of each of the ten data matrices.
Step 3: Take the first k left singular vectors of each decomposition.
Step 4: Compute the distance from probe to each subspace.
Step 5: Classify according to the smallest distance.

to achieve a cumulative energy of 97% for each. We then take k to be the maximum of these numbers ensuring that the singular vectors being used are capturing 97% of the energy of all the digit spaces. In practice, a threshold of 95% is used when one desires to design an efficient algorithm. When accuracy is the major concern, a threshold of 99% is used. Since we are concerned with both efficiency and accuracy, we chose the midpoint of these two thresholds, i.e., 97%. If our images are size $m \times n$ and we used k left singular vectors for each subspace, then calculating the quantity $||U_k U_k^t P - P||_2$ requires $4mnk + 2mn - k$ flops for each of the ten digit spaces. Notice that this total does not include the flops for a computation of SVD. This is due to the face that the SVD of the digit spaces is performed in the training phase of the algorithm. The basis vectors are pre-computed and then stored for later use.

CHAPTER 3

TANGENT DISTANCE

Introduction

In the SVD approach described above, we assumed that our digits lied in a subspace and, in turn, that our data was linear. A more general approach is to assume that it is nonlinear. In this section, we will treat each digit as a point on a high dimensional manifold. The question now becomes how to calculate the distance from a probe to each of these manifolds. A very simple and naive approach would be to compute the Euclidean distance to each digit in the training set and classify the probe as the digit yielding the shortest distance. This is a type of Nearest Neighbor Algorithm. Since this is essentially a pixel by pixel comparison of two images, the accuracy of the algorithm is highly dependent on the size of the training data. We would need our training set to include an image of almost every variation of each digit. For example, not only would we need an image of the number '7' in our training set, but we would also need an image of a '7' that has been shifted to the left/right and up/down. This is clearly unfeasible and furthermore, it also has the makings of a very inefficient algorithm since we would have to compare to every images. Thus, we need a measure of distance that is invariant to small transformations of the digits. The reason we say small transformations is that large transformations may alter a digit too much. For example, if an image of '6' is rotated a full 180^o we obtain a '9' [1]. Again, let us treat an image P as a vector in $\mathbb{R}^{784}$. If we assume that the set of allowable transformations (e.g. translation, scaling, rotation) are continuous, then applying

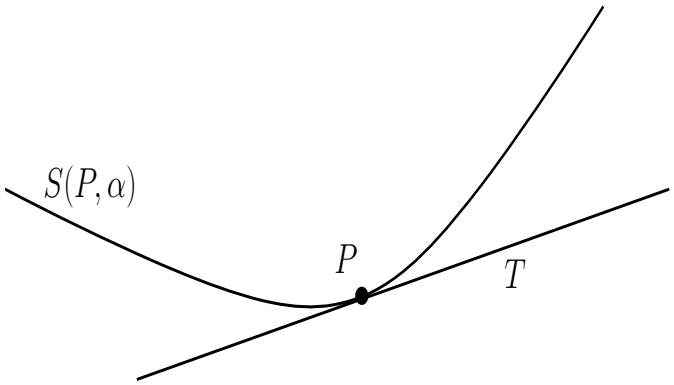


FIGURE 8. Approximation of surface by its tangent plane.

one or a combination of these transformation to the vector P will result in a surface in $\mathbb{R}^{784}$. As an example, suppose that the only transformation allowed is rotation which is dependent on a parameter α. If P is transformed according to the transformation $s(P, \alpha)$, then the set of all transformations

$$S_P = \{x \mid \exists\ \alpha \text{ for which } x = s(P, \alpha)\} \tag{3.1}$$

is a one dimensional curve in $\mathbb{R}^{784}$. In general, if the we allow n transformations, α will be an n-dimensional vector and S_P will be an n-dimensional surface. It is easy to see that $s(P, 0) = P$ (i.e. applying no transformations to P will not change P). For the purpose of this study, we will assume that $s(P, \alpha)$ is differentiable, ensuring that S_P is a smooth surface. This is of upmost importance since the heart of the algorithm relies on the ability to find the tangent plane.

Although we now have a manifold, finding the distance from a probe to this structure would be very expensive and unreliable since the manifold in nonlinear [1]. To remedy this, we will approximate the surface by its tangent plane. In Figure 8 we see an illustration of the tangent plane T at a point P on the manifold generated by the transformations, $s(P, \alpha)$.

The problem then reduces to finding the distance to a plane which is much simpler to solve. If we take the Taylor expansion of $s(P, \alpha)$ around $\alpha = 0$:

$$s(P, \alpha) = s(P, 0) + \alpha \frac{\partial s(P, \alpha)}{\partial \alpha} + O(\alpha^2) \approx P + L_P \alpha \tag{3.2}$$

where $L_P = \frac{\partial s(P,\alpha)}{\partial \alpha}$, we see that the linear approximation of the surface is completely characterized by P and L_P. Now, instead of simply calculating the Euclidean distance from a point $E \in \mathbb{R}^{784}$ to P, we can calculate the distance from E to the tangent plane at P; this is what is called a *One-Sided Tangent Distance*. There is also a *Two-Sided Tangent Distance* where the tangent plane for E is also calculated and we find the distance between the two planes. This is the measure that we will use in the algorithm. In Figure 9, we have an illustration of the two-sided tangent distance and the Euclidean distance between two points E and P whose tangent planes are denoted T_E and T_P, respectively [1].

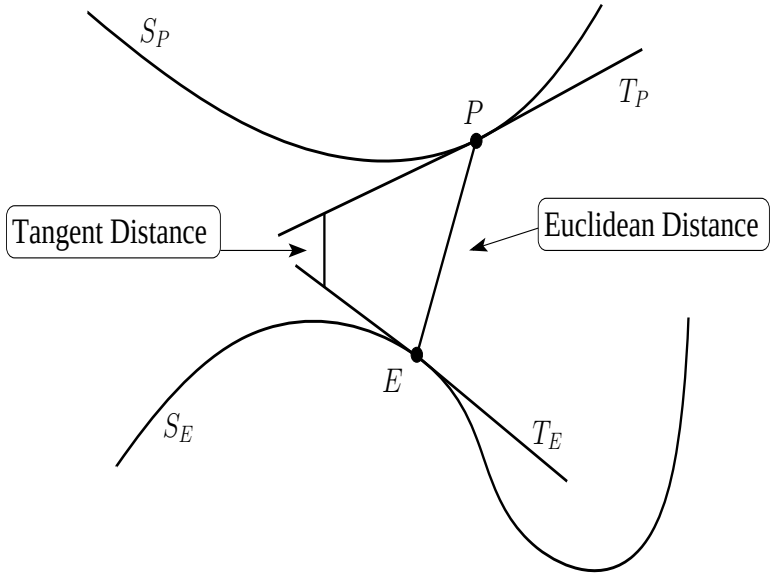


FIGURE 9. Two-sided tangent distance and Euclidean distance.

Implementation

At this point, we have two immediate questions that we need to answer: (1) How do we find the tangent plane and (2) how do we compute the distance between the two planes. The details of finding the tangent plane are a bit complicated and thus, we will save that discussion for the next section. In this section, we will concern ourselves with question (2) assuming that we have a way of finding tangent vectors. The derivation that follows was first presented in [1]. Suppose that we allow a total of m possible affine transformations (e.g. rotations, translations, scalings, etc.). Then the transformation parameter α is a vector in $\mathbb{R}^m$, $\alpha = (\alpha_1, \ldots, \alpha_m)$. Therefore, our tangent plane at P will be m-dimensional and generated by the columns of the matrix L_P defined to be:

$$L_P = \left.\frac{\partial s(P,\alpha)}{\partial \alpha}\right|_{\alpha=0} = \left[\frac{\partial s(P,\alpha)}{\partial \alpha_1}, \ldots, \frac{\partial s(P,\alpha)}{\partial \alpha_m}\right]_{\alpha=0} \tag{3.3}$$

In other words, the columns of L_P form a basis for the tangent plane at P. Moreover, they are the vectors tangent to the manifold. Suppose we have two images, E and P, with tangent planes T_E and T_P, respectively. Then the two-sided tangent distance between the two is defined to be:

$$TD(E,P) = \min_{x \in T_E, y \in T_P} ||x - y||_2^2 \tag{3.4}$$

The equations for the tangent planes T_E and T_P are given by

$$E^{'}(\alpha_E) = E + L_E \alpha_E \tag{3.5}$$

$$P^{'}(\alpha_P) = P + L_P \alpha_P \tag{3.6}$$

where L_E and L_P are, as before, the matrices whose columns are the tangent vector at E and P, respectively. Substituting these into Equation(3.4), we obtain

$$TD(E,P) = \min_{\alpha_E,\alpha_P} ||E^{'}(\alpha_E) - P^{'}(\alpha_P)||^2 \tag{3.7}$$

To solve this minimization problem, we can differentiate the expression $||E^{'}(\alpha_E) - P^{'}(\alpha_P)||^2$ with respect to α_E and α_P and set each expression equal to zero:

$$2(E^{'}(\alpha_E) - P^{'}(\alpha_P))^T L_E = 0 \tag{3.8}$$

$$2(E^{'}(\alpha_E) - P^{'}(\alpha_P))^T L_P = 0 \tag{3.9}$$

Substituting equations (3.5) and (3.6) into these expressions we obtain

$$L_P^T(E - P - L_P\alpha_P + L_E\alpha_E) = 0 \tag{3.10}$$

$$L_E^T(E - P - L_P\alpha_P + L_E\alpha_E) = 0 \tag{3.11}$$

Solving for α_E and α_P respectively gives

$$(L_{PE}L_{EE}^{-1}L_E^T - L_P^T)(E - P) = (L_{PE}L_{EE}^{-1}L_{EP} - L_{PP})\alpha_P \tag{3.12}$$

$$(L_{EP}L_{PP}^{-1}L_P^T - L_E^T)(E - P) = (L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \tag{3.13}$$

where $L_{EE} = L_E^T L_E$, $L_{PE} = L_P^T L_E$, $L_{EP} = L_E^T L_P$, and $L_{PP} = L_P^T L_P$. Both Equation (3.12) and (3.13) are linear systems that can be solved by Gaussian elimination to obtain α_E and α_P. Plugging these vectors back into (3.7) yields the tangent distance between the two patterns E and P. If we insert the formulas for $E^{'}(\alpha_E)$ and $P^{'}(\alpha_P)$ into (3.7), we get

$$TD(E,P) = \min_{\alpha_E,\alpha_P} ||E + L_E\alpha_E - P - L_P\alpha_P|| \quad (3.14)$$

$$= \min_{\alpha_E,\alpha_P} ||(E-P) + L_E\alpha_E - L_P\alpha_P|| \quad (3.15)$$

Simard warns that Equation (3.4) is singular if some of the tangent vectors of E and P are parallel [1]. While this is unlikely, it is possible that a pattern could repeat in the training and testing set. To remedy this, one can introduce a spring constant k and replace Equation (3.4) by

$$TD(E,P) = \min_{x\in T_E, y\in T_P} ||E + L_E x - P - L_P y||_2^2 + k||L_E x||_2^2 + k||L_P y||_2^2 \quad (3.16)$$

Going through the derivation as before we get

$$(L_{PE}L_{EE}^{-1}L_E^T - (1+k)L_P^T)(E-P) = (L_{PE}L_{EE}^{-1}L_{EP} - (1+k)^2 L_{PP})\alpha_P \quad (3.17)$$

$$(L_{EP}L_{PP}^{-1}L_P^T - (1+k)L_E^T)(E-P) = ((1+k)^2 L_{EE} - L_{EP}L_{PP}^{-1}L_{PE})\alpha_E \quad (3.18)$$

which will always have a solution for $k > 0$. The physical interpretation of this, which can be seen in Figure 10, is that P is attached to a point in its tangent plane, x, and E is attached to a point in its tangent plane, y, by a spring of elasticity k while x and y are attached by a spring of elasticity 1. The new "spring" tangent distance is the total potential energy stored in all three springs at equilibrium.

As was mentioned earlier, we would like our metric to be invariant to small transformations. In Figure 11 we have taken an image of the digit '3' and translated it ten pixels to the left and right. We then take one image of every digit $0, 1, \ldots, 9$ as our training set. In Figure 11(a), we calculate the Euclidean distance

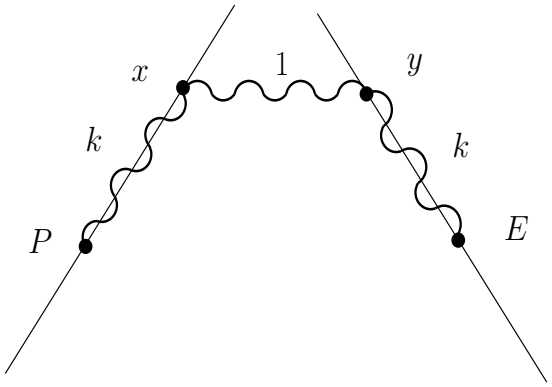
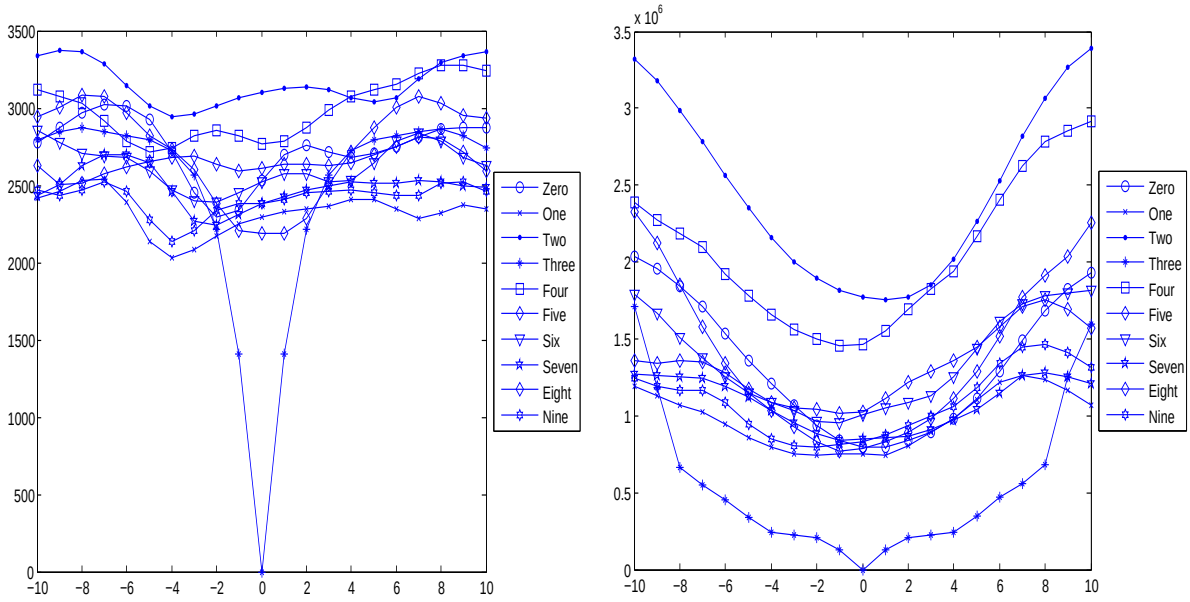


FIGURE 10. Physical interpretation of spring tangent distance.

from every translated '3' to each of the digit in the training set and plot the distances. Thus, every curve is a plot of the Euclidean distance between a digit in the training set and the translated '3's. The V-shaped curve is the distance from the translated '3's to the '3' in the training set. If we classify the translated digits according to smallest distance, the Euclidean metric only classifies translations up to ± 2 pixels correctly, that is, two pixels to the left and right. In Figure 11(b), we have done the same except that we have used the spring tangent distance with $k = .05$ instead of the Euclidean distance. Notice that the tangent distance can accurately recognize the digit up to translations of about ± 9 pixels which is a drastic improvement over Euclidean distance. Similar behavior can be seen when the digit '3' is translated vertically, rotated, and scaled which demonstrates the invariance of the Tangent Distance metric under these transformations. The plots for these transformations look very similar to that in Figure 11, but are not included in order to avoid redundancy.

Calculating Tangent Vectors

In this section we address the question of how to obtain the tangent vectors needed in the calculation of the tangent distance. While there are multiple



(a) Euclidean Distance (b) Tangent Distance

FIGURE 11. Comparison of Euclidean distance and tangent distance.

methods of obtaining tangent vectors, we take an approach very similar to that in Chapter 1 and was motivated by the work in [5]. For a method involving numerical differentiation, the reader is encouraged to see [6]. The key to our method is the observation that a tangent plane is essentially a subspace except for the fact that a tangent plane does not necessarily contain the origin. However, this can easily be remedied by shifting the point of tangency to the origin. Then the problem of finding tangent vectors is identical to the task of finding basis vectors for a subspace which we accomplished in Chapter 1 using the left singular vectors from the singular value decomposition and an energy criterion. Thus, we can obtain tangent vectors for a point P on the manifold S_P by the following steps: (1) Find points in S_P near P and place them in the columns of a data matrix. (2) Subtract P from every column. (3) Take the SVD of the data matrix. (4)

Determine how many singular values are needed to capture 97% of the energy; the corresponding number of left singular vectors will be the tangent vectors. To illustrate this method, let us consider the toy example of finding the tangent vectors of the sphere at the north pole, i.e. $P = (0, 0, 1)$. We generated 1500 points on the unit sphere and took those within a distance of .1 from P. As we can see in Figure 12, the method yields a fairly good approximation.

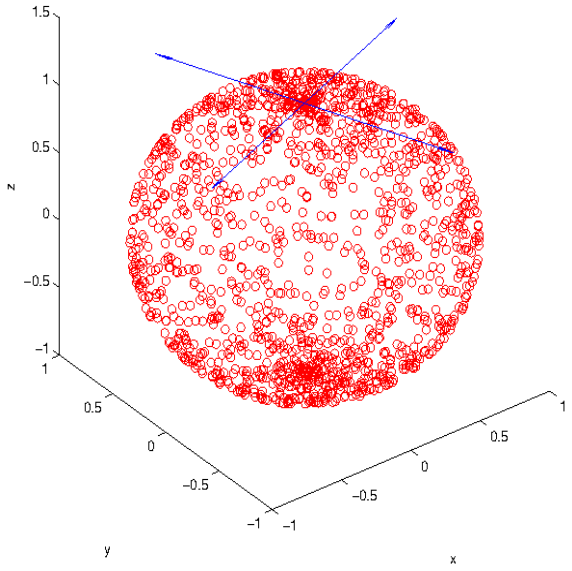


FIGURE 12. Tangent vectors at the north pole of the unit sphere.

One issue that arises when applying this method is that the MNIST database is a bit sparse making it difficult to find data points near a particular image. Thus, for the purpose of this study, we created neighboring points using ten vertical and ten horizonal translations from -5 to 5 pixels, ten rotations from $\frac{-\pi}{8}$ to $\frac{\pi}{8}$, and ten scalings from .8 to 1.2. Figure 13 shows the tangent vectors we obtain for a '2' in MNIST when using our method with the neighborhood described. While it is unclear what some of these vectors represent, the affine transformations used to form the neighborhood around the '2' can be seen in the

first row of Figure 13. Consider the first image from the left in the first row. The white pixels surrounding the black '2' indicate that this tangent vector represents scaling. The next image to the right represents vertical translation as is evidenced by the shadowing of white pixels above the '2'. Similarly, the third image from the left in the first row represents horizontal translation. Lastly the final image in the first row represents rotation.

FIGURE 13. Tangent vectors of a '2' in MNIST.

The algorithm is described in full in Algorithm 2 below. If our images are of resolution $m \times n$ and we form the neighborhoods around E and P using t transformations, the cost will be $27mnt$ for each neighborhood. Shifting each neighborhood to the origin requires mnt subtractions. Moreover, each neighborhood yields two data matrices of size $(mn) \times t$ whose SVD will cost approximately $4t^2(mn - \frac{1}{3}t)$ flops each [7]. If we utilize k_E and k_P tangent vectors for E and P, respectively, then solving for the tangent distance requires $mn(k_E + 1)(k_P + 1) + 3(k_E^3 + k_P^3)$ flops [1]. Thus, calculating the tangent distance

Algorithm 2 Tangent Distance Algorithm [1]

Input: Probe image P

Output: Classification of P as a digit $0, \ldots, 9$.

Step 1: For every training digit E, form the neighborhood around E and find the tangent vectors.

Step 2: For P, form its neighborhood and find the tangent vectors.

Step 3: Calculate the tangent distance from P to every digit in the training set.

Step 4: Classify P according to the smallest tangent distance calculated.

between a training digit E and a probe P requires a total of

$56tmn + 8t^2(mn - \frac{1}{3}t) + mn(k_E + 1)(k_P + 1) + 3(k_E^3 + k_P^3)$ flops.

CHAPTER 4

A GRASSMANNIAN APPROACH

Introduction

As was the case in Chapter 1, the underlying structure of the algorithm that will be presented in this chapter is a vector space. Let us begin with the familiar concept of angles.

Definition 4.1. *(Angle Between Two Vectors) Let u and v be vectors in $\mathbb{R}^n$. The angle between u and v is defined as*

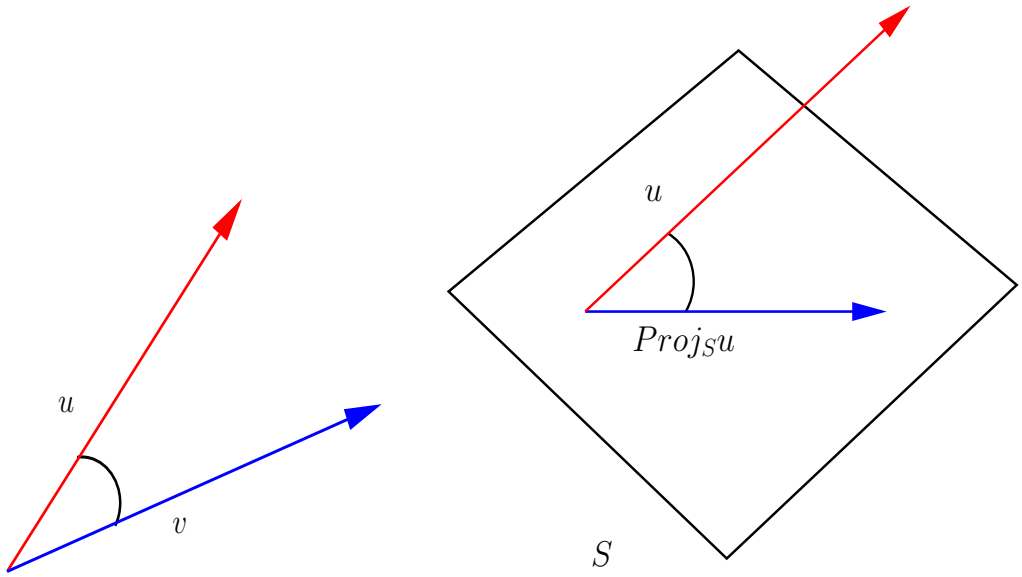
$$\angle(u,v) \;=\; \cos^{-1}\left(\frac{|v^t u|}{||u||_2||v||_2}\right).$$

Moreover, we can extend this definition to define the angle between a vector and a subspace.

Definition 4.2. *(Angle Between a Vector and a Subspace) Let u be a vector in $\mathbb{R}^n$ and S be a k-dimensional subspace of $\mathbb{R}^n$. Let $\mathrm{Proj}_S u$ denote the projection of u onto S. Then the angle between u and S is defined as*

$$\angle(u,S) \;=\; \cos^{-1}\left(\frac{|u^t \mathrm{Proj}_S u|}{||u||_2||\mathrm{Proj}_S u||_2}\right).$$

These are two familiar definitions that are easy to understand and visualize as we see in the illustrations in Figure 14. In 14(a) we have the angle between two vectors $u, v \in \mathbb{R}^n$ and (b) illustrates the angle between u and a subspace S.
For angles between a pair of subspaces, we consider the high-dimensional analog of vector angles, known as the *principal angles*.



(a) Angle between two vectors (b) Angle between a vector and a subspace.
FIGURE 14. The angle between two vectors and a vector and a subspace.

Definition 4.3. *[8] (Principal Angles) Let U and V be subspaces in $\mathbb{R}^n$ such that $p = dim(U) \geq dim(V) = q \geq 1$. Then the principal angles $\theta_k \in [0, \pi/2]$ for $k = 1, \ldots, q$ between U and V are defined recursively by*

$$\cos(\theta_k) = \max_{u \in U} \max_{v \in V} |u^t v| = |u_k^t v_k|$$

subject to $||u||_2 = ||v||_2 = 1$, $u^t u_i = 0$, and $v^t v_i = 0$ for $i = 1, \ldots, k-1$.

To explain this definition more thoroughly, suppose we are looking for θ_1. We must search through all of U and V to find the unit vectors that maximize the projection $|u^t v|$, or equivalently the vectors with the smallest angle between them. These vectors will be u_1 and v_1. To find θ_2, we again look for vectors in U and V to maximize the projection, but now our search is restricted to the orthogonal complement of u_1 and v_1 in U and V, respectively. Thus, in general, in order to find θ_k we must search in the orthogonal complements of $u_1, \ldots, u_{k-1}$ and

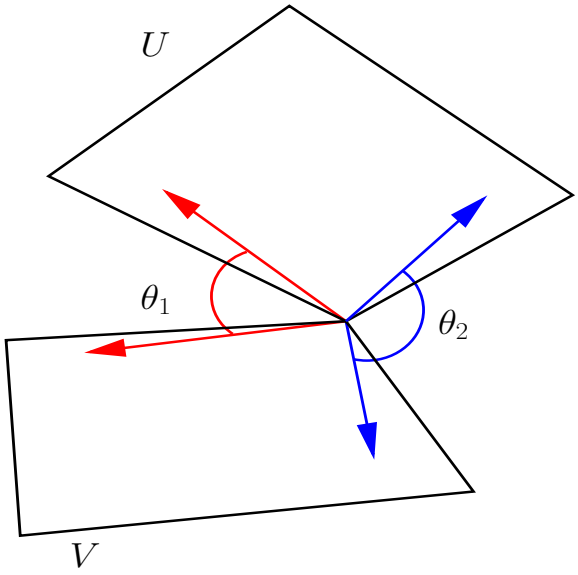


FIGURE 15. Illustration of the principal angles of two subspaces.

$v_1, \ldots, v_{k-1}$, respectively. Figure 15 illustrates the geometric interpretation of the principal angles between two subspaces in $\mathbb{R}^2$.

Given this recursive definition, one might think that finding the principal angles between two subspaces is difficult and computationally expensive. However, the task is greatly facilitated by the following theorem.

Theorem 4.4. *[8] Let U and V be subspaces in $\mathbb{R}^n$ such that $dim(U) = p$ and $dim(V) = q$, $p \geq q$. Assume that the columns of the matrices A and B form orthonormal bases for U and V, respectively. Let the SVD of the covariance matrix A^tB be*

$$A^tB \quad = \quad \tilde{U}\tilde{S}\tilde{V}^t$$

where $\tilde{S} = diag(\sigma_1, \sigma_2, \ldots, \sigma_q)$. Then the principal angles $\theta_1, \theta_2, \ldots, \theta_q$ associated with U and V satisfy

$$\cos\theta_k \quad = \quad \sigma_k, \quad k = 1, \ldots, q.$$

A proof of this result can be found in [8]. As an example, let us find the principal angles between the xy-plane and the yz-plane with orthonormal bases stored in $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$, respectively. The singular values of $A^tB = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$ are 1 and 0. Taking the inverse cosine of these values gives us the principal angles of 0 and $\frac{\pi}{2}$, respectively. This agrees with our intuition, given that the xy-plane and the yz-plane are orthogonal to each other and share one basis vector, i.e., [0 1 0]. Notice that we can also use principal angles to find the angle between two vectors and the angle between a vector and a subspace if we treat these scenarios as two 1-dimensional subspaces or a 1-dimensional subspace and a k-dimensional subspace, respectively. In these two situations (two vectors and a vector and a subspace), it is fairly straightforward to see how these structures can be used to form a digit classification algorithm. However, given a set of principal angles between two subspaces, it is unclear as to how to use them in designing a distance function. For this, let us introduce a structure known as the *Grassmann Manifold* or the *Grassmannian* which is where vector subspaces reside.

Definition 4.5. Grassmann Manifold. *The Grassmann Manifold, denoted $G(k, n)$, is the set of k-dimensional subspaces in $\mathbb{R}^n$,*

$$G(k, n) \quad = \quad \{W \subset \mathbb{R}^n \mid \dim(W) = k\}.$$

In Figure 16, we have an illustration of two k-dimensional subspaces U and V on the Grassmannian, $G(k, n)$.

Just as in traditional manifolds, we would like to find the distance between two points on the Grassmannian, i.e., the distance between two subspaces. It is

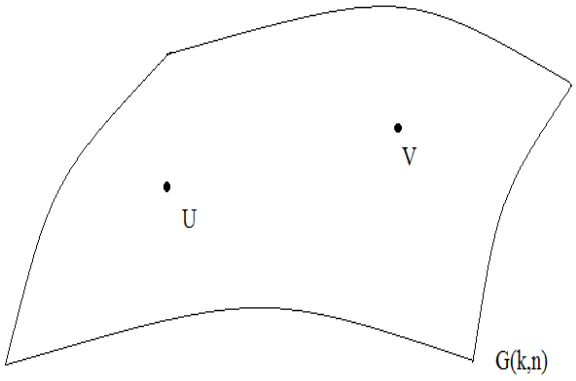


FIGURE 16. Illustration of two subspaces on the Grassmannian.

because of this that principal angles are so important since many metrics can be derived from them. Table 1 lists the ones that we will be considering in this study where θ is a vector containing the principal angles between a vector spaces U and V with $dim(U) = p \geq dim(V) = q$, i.e., $\theta = (\theta_1, \theta_2, \ldots, \theta_q)$ [9]. The derivation of these metrics can be found in [7].

TABLE 1. Table of Grassmannian Distances

Metric Name	Mathematical Expression
Fubini-Study	$d_{FS}(U,V) = cos^{-1}\left(\Pi_{i=1}^{q} cos\theta_i\right)$
Chordal 2-norm	$d_{c2}(U,V) = \|\|2sin\frac{1}{2}\theta\|\|_\infty$
Chordal F-norm	$d_{cF}(U,V) = \|\|2sin\frac{1}{2}\theta\|\|_2$
Geodesic (Arc Length)	$d_g(U,V) = \|\|\theta\|\|_2$
Chordal (Projection F-norm)	$d_c(U,V) = \|\|sin\theta\|\|_2$
Projection 2-norm	$d_{p2}(U,V) = \|\|sin\theta\|\|_\infty$

Implementation

The question we address in this section is to how we can apply these concepts of angles and metrics to design a successful algorithm. The idea of an

Algorithm 3 Vector to Vector
Input: Probe image P
Output: Classification of P as a digit $0, \ldots, 9$.
Step 1: For every digit in the training set, E, calculate the angle between E and P.
Step 2: Classify P according to the smallest angle.

angle between two vectors can very easily be implemented into an algorithm by simply calculating the angle between a probe and every digit in the training set, classifying the probe according to the smallest angle. When applying the other two concepts, we would like to maintain the nearest neighbor framework and thus, we will need to manually form a subspace around each digit. We can do this in a similar fashion as we did in Chapter 2, using rotations 10 from $-\frac{\pi}{8}$ to $\frac{\pi}{8}$, ten horizontal translations from -5 to 5 pixels, ten vertical translations from -5 to 5 pixels, and ten scalings from .8 to 1.2 to form a data matrix and then obtain a basis using QR decomposition or SVD. Algorithms 3, 4, and 5 provide a description of how the three algorithms will be implemented.

One thing to notice is that we are using QR decomposition to obtain a basis instead of SVD. The reason for this is that QR is computationally less expensive than SVD and we do not need the singular values of the data matrices for these implementations. We should also point out that we used all of the principal angles in the *Subspace to Subspace* algorithm to compute the distance between two subspaces.

Suppose our images are of size $m \times n$. Then Algorithm 3 is very efficient since calculating the angle between two vectors only requires approximately $6mn - 1$ flops. For Algorithms 4 and 5, suppose we generate t_E transformations

Algorithm 4 Vector to Subspace

Input: Probe image P

Output: Classification of P as a digit $0, \ldots, 9$.

Step 1: For every digit in the training set, form the data matrix using affine transformations.

Step 2: Find basis of data matrix using QR decomposition.

Step 3: Calculate the angle between P and the every subspace generated by a training digit.

Step 4: Classify P according to the smallest angle.

Algorithm 5 Subspace to Subspace

Input: Probe image P and Grassmannian metric

Output: Classification of P as a digit $0, \ldots, 9$.

Step 1: For every digit in the training set, E, form the data matrix using affine transformations.

Step 2: Find basis of data matrix using QR decomposition.

Step 3: Form data matrix for P using affine transformations.

Step 4: Find basis of P's data matrix using QR decomposition.

Step 5: Calculate the distance between the subspace generated by P and every subspace generated by a training digit.

Step 6: Classify P according to the smallest distance.

about the training digit E and t_P about the probe P. This will require $27mnt_E$ and $27mnt_P$, respectively. In order two find the principal angles, we need two reduced QR decompositions and a reduced SVD ($\frac{2}{3}t_E^2mn + \frac{2}{3}t_P^2mn + 4t_E^3$) and a multiplication of matrices of size $t_E \times mn$ and $mn \times t_P$ ($t_E t_P(2mn - 1)$) [7]. Thus, for Algorithm 4 we have $t_E = t$ and $t_P = 1$ giving us a total of $4t^3 + \frac{2}{3}t^2mn + 29tmn + \frac{83}{3}mn - t$ flops. Algorithm 5, with $t_E = t_P = t$, requires $4t^3 + \frac{10}{3}t^2mn + 54tmn - t^2$ flops.

Figure 17 gives us a visualization of Algorithm 4 via a flow chart where $x_1, x_2, \ldots, x_n$ are the images in the training set and $U_1, U_2, \ldots, U_n$ are the corresponding subspaces that were manually generated from the transformations. The bottom row illustrates the probe, P, being projected onto each subspace to find the principal angle for classification.

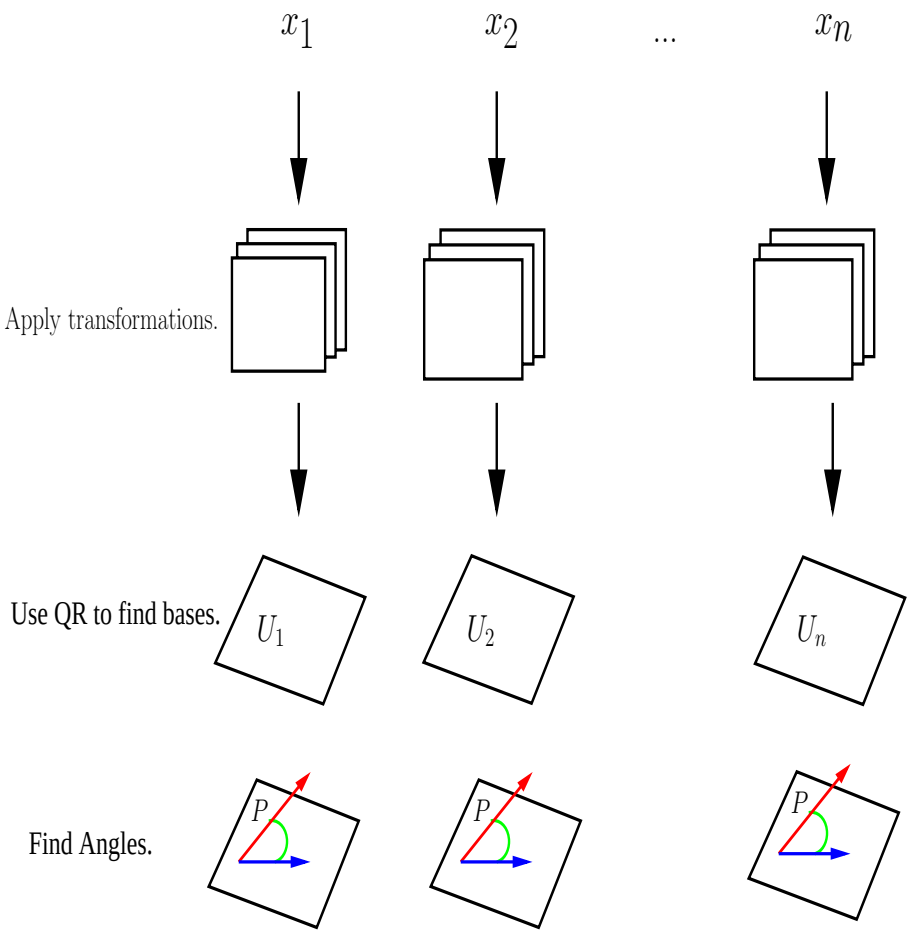


FIGURE 17. Flow chart for vector to subspace algorithm.

CHAPTER 5

EXPERIMENTS AND RESULTS

In this chapter we present the results of the algorithms that were described in Chapters 2, 3, and 4 on the MNIST database. While the SVD-based algorithm is very efficient, those that were presented in Chapters 2 and 3 are not, thus making it impractical to test on the entire database. Instead, what we have done here is to randomly select 50 images from the training set of each digit to form a smaller training set. We then tested the algorithm on the first 50 images in the testing set of each digit. In total, we are using 500 images from the training set and testing against 500 from the testing test. The measurement used to determine the effectiveness of the algorithms was *Classification Rate* (CR) which is defined as

$$\text{Classification Rate} \;=\; \frac{\text{Number of True Positives}}{\text{Number of Classifications}}$$

We ran each algorithm 10 times, each time using a different set of images for training, but the same testing set. In Table 2, we have the average CR of each algorithm along with its standard deviation and the time it takes to classify one digit. The algorithms were executed on a CPU with a speed of 2GHz and 1GB of RAM.

One thing that should be pointed out is that despite its relatively poor performance, the tangent distance algorithm is one of the most successful methods for classifying digits [1]. However, more is needed to make this method work. For example, it is strongly recommended in [1] and [6] that one smooth the images with a Gaussian before applying the method. Also, one should look at

TABLE 2. Results of Algorithms on MNIST Database

Algorithm	Average CR	Standard Deviation	Time(s)
SVD [3]	87.74%	1.95%	0.0007
Tangent Distance [1]	80.36%	2.06%	0.5749
Vector to Vector	82.02%	1.27%	0.0180
Vector to Subspace	90.07%	0.80%	0.0305
Fubini-Study	42.12%	1.94%	0.7134
Chordal 2-norm	51.20%	1.24%	0.7134
Chordal F-norm	82.00%	1.54%	0.7134
Geodesic (Arc Length)	81.84%	1.62%	0.7134
Chordal (Projection F-norm)	82.48%	1.46%	0.7134
Projection 2-norm	51.72%	1.68%	0.7134

transformations beyond the standard set of translations, rotations, and scalings. In [1], transformations such as thickening are considered while [5] looks at transformations inherent to the data. For more ways of improving the algorithm, the reader is encouraged to read [1].

Notice that the classification rate of the Subspace to Subspace algorithm is highly dependent on the norm used. The Chordal norm, the most successful of the norms, achieved a classification rate of 82.48% while the worst norm, Fubini-Study, failed to reach a classification rate of 50%.

Overall, our Subspace to Subspace methods did not perform as well as expected, but our Grassmannian approach proved to be very effective in our Vector to Subspace method which achieved the highest classification rate of 90.07% and the lowest standard deviation of 0.80%. The classification rate is almost 3% higher

than the next closest algorithm, the SVD-based method. It should be pointed out that in order to achieve the times in Table 2 we performed several computations off-line. In the SVD-based method, the bases for the ten digit spaces were pre-computed and stored for later use. In the Tangent Distance algorithm, we pre-computed the tangent vectors of the training digits. Lastly, in the Vector to Subspace and Subspace to Subspace algorithms, the manually generated subspaces of the training digits (e.g. the transformations and QR decompositions) were computed off-line. One might feel inclined to use the SVD-based method over the Vector to Subspace due to its efficiency, but with the constant improvements of computer hardware resulting in faster processors, one should focus more on the classification rate of the algorithm to determine its effectiveness.

CHAPTER 6

SUMMARY AND CONCLUSIONS

In this thesis, we reviewed two geometry based algorithms for the handwritten digit recognition problem while introducing three novel algorithms based on the Grassmannian manifold. One of our three algorithms, Vector to Subspace, achieved the highest classification rate in this study while simultaneously achieving one of the fastest times.

While the Vector to Subspace algorithm bears a resemblance to the SVD-based algorithm, there are two key differences that make this algorithm more effective. Firstly, the Vector to Subspace method makes significantly more comparisons. The SVD-based algorithm compares an unknown digit with ten subspaces while the Vector to Subspace method compares the digit to 500 subspaces, i.e. the size of the training set. And second, the manually generated subspaces exhibit more linear behavior than the subspaces in the SVD-based method. The transformations (e.g. rotations, translations, and scalings) of a training digit are much closer to the digit than other digits in the training set. Since smooth surfaces are *locally* linear, this results in a better linear approximation.

The idea of a manually generated subspace arose from one of the problem we faced when working with the MNIST database. Compared to other databases, such as NIST, MNIST is relatively sparse making it very difficult to find neighboring digits. Because of the spareness, we were unable to use the method described in [5] to find tangent vectors.

Given its success in this study, we plan on testing the Vector to Subspace method on the entire MNIST database and comparing its results with other algorithms, including learning based algorithms, that have been shown to be the most successful on MNIST. Moreover, we have begun to investigate ways of improving the efficiency of the algorithm. In particular, we are looking into using the Karcher Mean to obtain a single subspace representative for each digit. This will make the time it takes to classify one digit comparable to that of the SVD-based method.

APPENDIX

APPENDIX A

MATLAB CODES

Code #1 This code tests the SVD-based algorithm presented in Chapter 1 where each set of training digits is treated as subspace and the bases vectors are obtained by using the Singular Value Decomposition and an energy criterion. The unknown digit P is then classified according to the smallest residual.

```
clear all
load digits

%Number of training digits to be used
numTrain = 50;

%Choose 50 random indices
trainIndices = randperm(5000);
trainIndices = trainIndices(1:numTrain);

%Number of digits to be tested
numTest = 50;

%Find maximum energy
digitEnergy = zeros(1,10);

for i = 1:10
    S = svd(trainDigits(:,trainIndices,i),0);
    digitEnergy(i) = cumEnergy(S);
end

%Number of singular vectors to be used
numSingVec = max(digitEnergy);

%Array to store left singular vectors
singVec = zeros(784,numSingVec,10);

%Take SVD of data and keep desired left singular vectors
for i = 1:10
    [singVec(:,:,i),S,V] = svds(trainDigits(:,trainIndices,i),
                                numSingVec);
end

counter = zeros(1,10);

for i = 1:10

    for j = 1:numTest
```

```
        %Probe
        P = testDigits(:,j,i);

        %Array to store distances
        D = zeros(1,10);

        %Find distance from probes to digit spaces.
        for k = 1:10
            D(k) = norm(singVec(:,:,k)*((singVec(:,:,k)')*P) - P);
        end

        %Smallest distance
        m = min(D);

        %Count how many places the minimum occurs
        minCount = 0;
        for k = 1:10
            if D(k) == m
                minCount = minCount + 1;
            end
        end

        if minCount == 1     %Minimum only occurs once
            if D(i) == m   %Minimum occurs in right position
                %Correct classification; update counter
                counter(i) = counter(i) + 1;
            end
        end
    end
end

accuracy = sum(counter)/(10*numTest);
```

Code #2 This function determines the number of singular values needed to achieve an energy of at least 97%. The input must be a row or column vector of singular values in decreasing order.

```
function n = cumEnergy(D)

totalE = sum(D.^2);

n = 1;
```

```
partialE = 0;

while true

    partialE = partialE + D(n)^2;

    if partialE/totalE > .97
        break
    else
        n = n + 1;
    end
end
```

Code #3 This code tests the Tangent Distance algorithm presented in Chapter 3 where a neighborhood is formed around the test digit E and the probe P using translations, rotations, and scalings and the tangent vectors are obtained by using the SVD and an energy criterion. The spring tangent distance is then calculated with $k = .05$.

```
clear all
load digits

%Number of training digits to be used
numTrain = 50;

%Choose random indices
trainIndices = randperm(5000);
trainIndices = trainIndices(1:numTrain);

%Number of images to be tested for each digit
numTest = 50;

%counter
c = zeros(10,1);

%Arrays to store tangent distance for each digit
D = zeros(numTrain,10);

for k = 1:10

    for j = 1:numTest

        %Get Probe
```

```
        P = testDigits(:,j,k);

        %Get Tangent Vectors for P
        Ball_P = transformations(P);
        [UP,SP,TP] = svd(Ball_P - repmat(P,1,size(Ball_P,2)),0);
        P_energy = cumEnergy(diag(SP));
        LP = UP(:,1:P_energy);

        %Calculate distance to each digit in training set
        for i = 1:length(trainIndices)

            for n = 1:10

          %Get Training Digits
E = trainDigits(:,trainIndices(i),n);

                %Get Tangent Vectors for E
                Ball_E = transformations(E);
                [UE,SE,TE] = svd(Ball_E
                                 - repmat(E,1,size(Ball_E,2)),0);
                E_energy = cumEnergy(diag(SE));
                LE = UE(:,1:E_energy);

                D(i,n) = springtangentDistance(P,LP,E,LE,.05);
            end
        end

        %Minimum distance for every digit
        minDists = min(D);

        %Find where minimum distance occurs
        [C,I] = min(minDists);

        %Count number of times minimum occurs
        minCount = 0;
        for i = 1:10
           if minDists(i) == C
                minCount = minCount + 1;
            end
        end

        if minCount == 1 %Minimum occurs only once
```

```
            if I == k %Minimum occurs at correct digit
                %Correct classification; update counter
                c(k) = c(k) + 1;
            end
        end
    end
end

accuracy = sum(c)/(10*numTest);
```

Code #4 This function translates a gray scale images in the x and/or y direction.

```
function x = translate(image,tx,ty)

    %Get size of images
    A = image;
    [n,m] = size(A);

    %Construct rotation matrix
    T = [1, 0, tx;
        0, 1, ty;
        0, 0, 1];

    [X,Y] = meshgrid(1:n,1:m);
    X1 = [X(:)]';
    Y1 = [Y(:)]';

    %Form matrix to store coordinates
    xy_mat = vertcat(X1, Y1, ones(1,n*m));

    %Apply transformation to coordinates
    Z = inv(T)*(xy_mat);

    %New Coordinates
    XI = Z(1,:);
    YI = Z(2,:);

    %Reshape coordinates to have same form as X and Y
    XI = reshape(XI', n, m);
    YI = reshape(YI', n, m);

    %Interpolate values
```

```
    final = interp2(X,Y,A,XI,YI);

    %Set background pixels to white
    for i = 1:n
        for j = 1:m
            if isnan(final(i,j)) == 1
                final(i,j) = 255;
            end
        end
    end

    x = final;

end
```

Code #5 This function rotates a gray scale images about its center.

```
function x = rotate(image,theta)

    %Get size of images
    A = image;
    [n,m] = size(A);

    %Translation Matrix to ensure rotation is done about center
    T = [1, 0, (-m/2);
        0, 1, (-n/2);
        0, 0, 1];

    %Construct rotation matrix
    R = [cos(theta), -sin(theta), 0;
        sin(theta), cos(theta), 0;
        0, 0, 1];

    [X,Y] = meshgrid(1:n,1:m);
    X1 = [X(:)]';
    Y1 = [Y(:)]';

    %Form matrix to store coordinates
    xy_mat = vertcat(X1, Y1, ones(1,n*m));

    %Apply transformation to coordinates
    Z = (inv(T)*R*T)*(xy_mat);
```

```
    %New Coordinates
    XI = Z(1,:);
    YI = Z(2,:);

    %Reshape coordinates to have same form as X and Y
    XI = reshape(XI', n, m);
    YI = reshape(YI', n, m);

    %Interpolate values
    final = interp2(X,Y,A,XI,YI);

    %Set background pixels to white
    for i = 1:n
        for j = 1:m
            if isnan(final(i,j)) == 1
                final(i,j) = 255;
            end
        end
    end

    x = final;
end
```

Code #6 This function scales a gray scale image about a point in the image plane (tx, ty) by a factor of sx in the x direction and sy in the y direction.

```
function x = scale(sx,sy, image, tx,ty)

    A = image;
    [n,m] = size(A);

    %Construct scaling matrix
    M = [sx, 0, -tx*(sx-1); 0, sy, -ty*(sy-1); 0, 0, 1];

    [X,Y] = meshgrid(1:n,1:m);
    X1 = [X(:)]';
    Y1 = [Y(:)]';

    %Form matrix to store coordinates
    xy_mat = vertcat(X1, Y1, ones(1,n*m));

    %Apply transformation to coordinates
    Z = inv(M)*(xy_mat);
```

```
    %New Coordinates
    XI = Z(1,:);
    YI = Z(2,:);

    %Reshape coordinates to have same form as X and Y
    XI = reshape(XI', n, m);
    YI = reshape(YI', n, m);

    %Interpolate values
    final = interp2(X,Y,A,XI,YI);

    %Set background pixels to white
    for i = 1:n
        for j = 1:m
            if isnan(final(i,j)) == 1
                final(i,j) = 255;
            end
        end
    end

    x = final;
end
```

Code #7 This function performs various transformations of an image (rotations, translations, scalings). Input image must be a column vector that can be reshaped into a square image.

```
function [trans] = transformations(P)

trans = [];
nsquared = size(P,1);
n = sqrt(nsquared);

%Reshape image
I = reshape(P,n,n);

%Number of transformations
k = 10;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Rotations                        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
a = -pi/8;
b = pi/8;

%Step Size
h = (b-a)/k;

for i = 1:k
    temp = rotate(I,a+(h*i));
    trans = [trans temp(:)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%x-translations                 %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = -5;
b = 5;

%Step size
h = (b-a)/k;

for i = 1:k
    temp = translate(I,a+(h*i),0);
    trans = [trans temp(:)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%y-translations                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = -5;
b = 5;

%Step size
h = (b-a)/k;

for i = 1:k
    temp = translate(I,0,a+(h*i));
    trans = [trans temp(:)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%scalings                       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
a = .8;
b = 1.2;


%Step size
h = (b-a)/k;

for i = 1:k
    temp = scale(a+(h*i),a+(h*i),I,n/2,n/2);
    trans = [trans temp(:)];
end
```

Code #8 This function finds the Spring Tangent Distance between two digits. E and P are images that have been concatenated into column vectors. LE and LP are matrices whose columns are the the tangent vectors at E and P, respectively. Lastly, k is a nonnegative number. If $k = 0$, this calculates the standard tangent distance.

```
function std = springtangentDistance(E,LE,P,LP,k)

LEE = LE'*LE;
LEP = LE'*LP;
LPE = LP'*LE;
LPP = LP'*LP;

%Solve system

%alpha_P
Bp = (LPE*inv(LEE)*LE' - (1+k)*LP')*(E-P);
Ap = LPE*inv(LEE)*LEP - ((1+k)^2)*LPP;

alpha_P = Ap\Bp;

%alpha_E
Be = (LEP*inv(LPP)*LP' - (1+k)*LE')*(E-P);
Ae = ((1+k)^2)*LEE - LEP*inv(LPP)*LPE;

alpha_E = Ae\Be;

Pprime = P + LP*alpha_P;
Eprime = E + LE*alpha_E;

std = (norm(Eprime - Pprime)^2)
```

```
          + k*(norm(LE*alpha_E)^2)
          + k*(norm(LP*alpha_P)^2);
```

Code #9 This codes tests the Vector to Vector algorithm presented in Chapter 4 in which the angle between a training digit E and a probe P is used as the means of classification.

```
clear all
load digits

%Number of images to be used in test set
%per digit
numTrain = 50;
trainIndices = randperm(5000);
trainIndices = trainIndices(1:numTrain);

%Number of images to be tested for
%every digit
numTest = 50;

%Array to store distances
dist = zeros(numTrain,10);

%Counter
c = zeros(1,10);

for k = 1:10

        for i = 1:numTest

            P = testDigits(:,i,k);

            for j = 1:length(trainIndices)

                for n = 1:10

                    E = trainDigits(:,trainIndices(j),n);


                    %Calculate Angle
                    angle = acos((P'*E)/(norm(P)*norm(E)));

                    dist(j,n) = angle;
```

```
            end
        end

        digitMins = min(dist);

        m = min(digitMins);

        %Count how many places the minimum occurs
        minCount = 0;
        for l = 1:10
            if digitMins(l) == m
                minCount = minCount + 1;
            end
        end

        %Minimum only occurs once
        if minCount == 1
            %Minimum occurs in right position
            if digitMins(k) == m
                %Correct classification; update counter
                c(k) = c(k) + 1;
            end
        end
    end
end

accuracy = sum(c)/(10*numTest);
```

Code #10 This codes tests the Vector to Subspace algorithm presented in Chapter 4 where a subspace is manually formed around a training digit E and P is classified according to the smallest angle it makes with each subspace.

```
clear all
load digits

%Number of images to be used in test set
%per digit
numTrain = 50;
trainIndices = randperm(5000);
trainIndices = trainIndices(1:numTrain);

%Number of images to be tested for
```

```
%every digit
numTest = 50;

%Array to store distances
dist = zeros(numTrain,10);

%Counter
c = zeros(1,10);

for k = 1:10

        for i = 1:numTest

            P = testDigits(:,i,k);


            for j = 1:length(trainIndices)

                for n = 1:10

                    E = trainDigits(:,trainIndices(j),n);
                    Ball = transformations(E);

                    %Calculate Principal Angle
                    angle = principalAngles(P,Ball);

                    dist(j,n) = angle;

                end
            end

            digitMins = min(dist);

            m = min(digitMins);

            %Count how many places the minimum occurs
            minCount = 0;
            for l = 1:10
                 if digitMins(l) == m
                     minCount = minCount + 1;
                 end
            end
```

```
            %Minimum only occurs once
            if minCount == 1
                 %Minimum occurs in right position
                 if digitMins(k) == m
                     %Correct classification; update counter
                     c(k) = c(k) + 1;
                 end
            end
        end
end

accuracy = sum(c)/(10*numTest);
```

Code #11 This code tests the Subspace to Subspace algorithm presented in Chapter 4 using the Chordal(Projection F-norm). Here, subspaces are manually formed around both the training digit E and the probe P. The distance between the two subspaces is then determined using the Chordal norm of the principal angles. The other 5 variations of this algorithm are implemented in exactly the same manner by simply changing the norm used on the principal angles.

```
clear all
load digits

%Number of images to be used in test set
%per digit
numTrain = 50;
trainIndices = randperm(5000);
trainIndices = trainIndices(1:numTrain);

%Number of images to be tested for
%every digit
numTest = 50;

%Array to store distances
dist = zeros(numTrain,10);

%Counter
c = zeros(1,10);

for k = 1:10

        for i = 1:numTest
```

```
    P = testDigits(:,i,k);

    Ball_P = transformations(P);

    for j = 1:length(trainIndices)

        for n = 1:10

            E = trainDigits(:,trainIndices(j),n);
            Ball_E = transformations(E);

            %Calculate Principal Angles
            angles = principalAngles(Ball_P,Ball_E);

            %Chordal(Projection F-norm)
            dist(j,n) = norm(sin(angles));

        end
    end

    digitMins = min(dist);

    m = min(digitMins);

    %Count how many places the minimum occurs
    minCount = 0;
    for l = 1:10
         if digitMins(l) == m
             minCount = minCount + 1;
         end
    end

    %Minimum only occurs once
    if minCount == 1
         %Minimum occurs in right position
         if digitMins(k) == m
             %Correct classification; update counter
             c(k) = c(k) + 1;
         end
    end
end
```

```
end

accuracy = sum(c)/(10*numTest);
```

Code #12 This function calculates the principal angles between two subspaces spanned by the columns of A and B.

```
function [angles] = principalAngles(A,B)

[Qa,Ra] = qr(A,0);
[Qb,Rb] = qr(B,0);

C = svd(Qa'*Qb,0);
angles = acos(C);
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] P. Simard, Y. L. Cun, J. Denker, and B. Victorri. "Transformation invariance in pattern recognition - tangent distance and tangent propagation." *Imaging System Technology*, vol. 11, pp. 181–194, 2001.

[2] Y. LeCun and C. Cortes. "The MNIST Database of Handwritten Digits." Internet: `http://yann.lecun.com/exdb/mnist/`, 1998 [Feb. 17, 2011].

[3] L. Elden. *Matrix Methods in Data Mining and Pattern Recognition*. SIAM, 2007.

[4] M. Kirby. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*. John Wiley & Sons, Inc., 2001.

[5] J.-M. Chang, M. Kirby, L. Krakow, J. Ladd, and E. Murphy. "Classification of images with tangent distance." Technical report, Colorado State University, 2004.

[6] B. Savas. "Analyses and test of handwritten digit algorithms." Master's thesis, Linköping University, 2002.

[7] J.-M. Chang. "Classification on the Grassmannians: Theory and Applications." Ph.D. thesis, Colorado State University, 2008.

[8] A. Björck and G. Golub. "Numerical methods for computing angles between linear subspaces." *Mathematics of Computing*, vol. 27(123), pp. 579–594, 1973.

[9] A. Adelman, T. Arias, and S. Smith. "The geometry of algorithms with orthogonality constraints." *Matrix Anal. Appl.*, vol. 20(2), pp. 303–353, 1999.

[10] J.-M. Chang, J. Beveridge, B. Draper, M. Kirby, H. Kley, and C. Peterson. "Illumination face spaces are idiosyncratic," presented at The International Conference on Image Processing, Computer Vision, & Pattern Recognition, Las Vegas, NV, 2006.