UNIVERSITY OF CALIFORNIA, SAN DIEGO

Optimizing the Distance Function for Nearest Neighbors Classification

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Ravi Mody

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I gratefully acknowledge Tim Marks for all his guidance and help during my research for this thesis. I also appreciate the helpful comments from Gary, Lawrence and Sanjoy while writing it.

ABSTRACT OF THE THESIS

Optimizing the Distance Function for Nearest Neighbors Classification

by

Ravi Mody
Master of Science in Computer Science

University of California, San Diego, 2009

Professor Garrison Cottrell, Chair

When working with high dimensional data, it is often essential to calculate the difference or “distance” between two points. One of the most common distance functions is the Euclidean distance; while this method is easy to implement, it often works poorly. We explore some of its deficiencies, and discuss how they have been overcome in previous research by taking advantage of statistics of the data and by using a priori information about the problem space.

We analyze two disparate methods that improve on Euclidean distance for classification. The first learns a Mahalanobis distance that is optimized on the statistics of the data to improve classification. The second incorporates a priori information using tangent distances to account for transformations that we intuitively know the classification should be invariant to. We combine these methods in a sequential manner that takes advantage of their unique strengths, improving the performance of either method by itself. Our combined method reduces the tangent distance's error on the USPS handwritten digit recognition dataset by 10.2% and the Mahalanobis distance method's error

by 44.6%.

Chapter 1

Introduction

When we want to find the distance between two objects, we usually pull out a tape measure and physically record the distance between them. This intuitively makes sense - objects that are further apart will show a larger distance on the tape measure. In machine learning, we often want a "tape measure" that exists in a higher dimensional space. Instead of two objects, we measure the distance between two pieces of data. And instead of seeking the physical amount of space between the two data points, we want to know how different they are. In other words, we seek an abstract tape measure, called the distance function, that measures the dissimilarity between data.

It is important to keep in mind that there is no "correct" distance function. A distance function that works well for one problem or dataset may work poorly on another. Therefore, we want to optimize the distance function for our problem, given what we know about its domain.

This is an important and well-studied problem in machine learning [Michalski, Stepp, and Diday, 1981] [Wilson and Martinez, 1997]. Still, many researchers continue to use the Euclidean distance, which is often a poor choice in practice. In this thesis, we look at some of the weaknesses of the Euclidean distance and explore ways to improve on it.

Application to Non-Parametric Learning

The distance function comes up frequently in non-parametric models of learning. A non-parametric model involves retaining the entire training dataset, and processing a test point by comparing it to the training set. This is in contrast to parametric methods, which throw out the training data after learning some parameters of a fixed model. Distance functions are important for non-parametric models because test data are usually classified by assuming they are more similar to training data that are close, i.e. with a small distance.

Therefore, we seek a distance function that in some way "optimally" brings points that we consider similar close, and pushes away points that we consider different. For example, if we have a collection of images of animals, a pair of images of dogs should ideally have a much smaller distance than a picture of a cat and a picture of a dog.

We look at the problem of classifying data using nearest neighbors, which gives a test point the same value as the nearest testing point. Obviously, a good distance measure is critical for good performance when using this simple method.

We approach this problem by combining two different improvements on Euclidean distance. The first improvement, called tangent distance, uses a priori knowledge of the problem to bring similar points closer. We often have some intuitive understanding of the problem domain that is difficult for current machine learning methods to learn. For example, we may know that when classifying whether a specific object exists in a picture, that object can exist anywhere in the image. This translational invariance is hard to learn without a very large dataset, but tangent distances allow us to approximately account for it quite easily. By building this knowledge into the distance measure, we can "teach" our model information about the problem that it probably could not have learned from the data alone.

In contrast, the second improvement, which we combine with the tangent distance, automatically learns statistics on the data that humans may have a difficult time realizing. The method involves learning a Mahalanobis distance function that can account for the interaction between different dimensions in the data. For example, a certain pixel in an image may heavily correlate with another pixel in a second image when the two images contain the same object, but not correlate when they contain different objects; this pattern can be incorporated into an improved Mahalanobis distance measure that is learned by various algorithms we look at.

The novel part of our method is how we combine the tangent distance with a Mahalanobis metric. Normally, in tangent distance, we transform two data points so similar points will be closer in Euclidean space, and then find the Euclidean distance between the transformed points. In our method, we bring the two similar data points closer in Euclidean space, and then apply the learned Mahalanobis metric between the transformed points. By doing this, we can combine our intuitive knowledge of the problem with a statistical learning algorithm, which we will show improves on either method alone.

Chapter 2

Background

The fundamental problem this thesis addresses is optimizing the distance function for classification using nearest neighbors. We only experimented with image data, but the process could be used for other problem types if suitable a priori transformations exist. In this chapter, we will first give background information that will help frame the problem, and then we will discuss the methods that our method is based on.

2.1 Classification

One of the most fundamental types of problem in machine learning is classification. In classification, all data are labeled into a finite number of groups; a data point, x is classified with its label (also called class) t, where $t \in C$. C is a finite set of possible labels. For example, in our experiments, we classify hand written digits, so $t_i \in \{0, 1 \ldots, 9\}$.

2.1.1 K Nearest Neighbors

K Nearest neighbors (KNN) is one of the simplest data classification algorithms that is widely used. Given a set of training data points, x_n, and their corresponding labels, t_n, we want to classify a test point, y. In KNN, we classify the test point with the labels of the training data that are closest to it (the nearest neighbors). The only parameter is k, which controls how many nearest neighbors of a point will be used for the classification.

Given a test point's k nearest neighbors, we can label it with a vote of the most common label from its neighbors. To break ties, various heuristics can be used, such as using the closest point, or an average of distances from each label.

The value of k can greatly influence the behavior of KNN. Small values of k search the space around a point more locally, whereas large values look further. Small values of k run the risk of falsely classifying a point in a set with noisy or quickly varying data. Large values of k essentially smooth the classification, but run the risk of averaging points that are far from the point. The optimal choice of k is therefore dependent on the density of the data. A good value of k can often be experimentally determined; in our experiments, we found $k = 1$ to work the best.

While often effective, KNN has some issues that should be considered. It requires the training data to be sampled evenly and densely enough to accurately cover the domain of possible test points. A hole in this sampling can cause test points to be classified ineffectively. As mentioned above, k should be chosen to reflect the sampling of data.

Another issue is how to work with image data when using nearest neighbors. We run into the problem that images are a 2-dimensional grouping of pixels, but KNN (and many other machine learning methods) work on vectors, which represent points in high dimensional space. We use the commonly accepted solution - simply vectorize the images. The order of the mapping from two dimensions to one dimension is not important, as long as it is consistent for each image.

The nearest neighbor classification can use any distance measure, including Euclidean, Mahalanobis, and tangent distance. The choice of distance function is critical to the performance of nearest neighbors, because the classification is based solely on what distances are returned. In the next section we give a more formal treatment to distance functions, and then discuss existing methods to optimize the function for nearest neighbors.

2.2 Distance Metrics

A distance metric (also called distance function or distance measure) is a function that takes two vectors as input, and returns a real positive number - the distance between the two vectors. Intuitively, in machine learning, we want the distance to indicate how “different” two vectors (or data points) are. The distance function should be small between similar data points, and large between dissimilar points. The mathematical definition of a distance function is any function that takes two inputs and follows the following rules, where $x, y, z \in \mathbb{R}^d$ [Munkres, 2000]:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0$ iff $x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

We will now discuss a few examples of distance metrics, and their properties.

2.2.1 Euclidean Distance

The Euclidean distance is perhaps one of the simplest and most intuitive distance measures. It is used often in machine learning, despite some weaknesses (which will be discussed below). One reason for its intuitiveness is that Euclidean geometry describes the 3-dimensional space we live in - a ruler measures the Euclidean distance between two objects.

The Euclidean distance is defined as

$$d(x, y) = \sqrt{(x-y)^\mathsf{T}(x-y)} \tag{2.1}$$

There are a few issues with Euclidean distance that make it a poor choice for many applications in machine learning. The first is the independence of dimensions in the calculation. We can rewrite 2.1 as

$$d(x, y) = \sqrt{\sum_{i=1}^{d} (x_{(i)} - y_{(i)})^2} \tag{2.2}$$

where $x_{(i)}$ is the i^{th} element of x. Notice that the distance is a sum of terms which are independent of each other, and that each term is only dependent on a single dimension. This can present some problems if the dimensions are not chosen carefully. For example, if we want to compare models of cars, we may choose one dimension to be the number of windows, and another dimension to be the number of doors. If, in every model of car, the number of windows and doors is the same, we are essentially double counting a single attribute of the car. This is usually not what we want. Euclidean distance has no way of taking the correlation between variables into account.

Another similar problem with Euclidean distance is the scaling of dimensions. If the only two attributes of cars we are looking at is weight in grams and number of doors, the weight of the car will dwarf the number of doors. In the typical car, the number of doors will essentially not be taken into account. Even changes in units (e.g. using pounds instead of kilograms) will change the meaning of the distance. We want a distance measure that can scale the dimensions independently.

The first two issues can be solved using a Mahalanobis metric, which will be discussed in section 2.2.2. A third issue is that the Euclidean distance does not allow us to use a priori knowledge about the problem space. In our experiments, we have a set of images with handwritten digits (0 to 9); we want to classify these by what by their digit. With a distance function and a labeled training dataset, we use nearest neighbors to classify the images. A problem arises due to the complexity of the problem, however. The number of possible images for even a single digit is huge. Each digit could be rotated, translated, stretched, etc. Each of these transformations drastically affects the Euclidean distance without changing what digit is written on the image. We

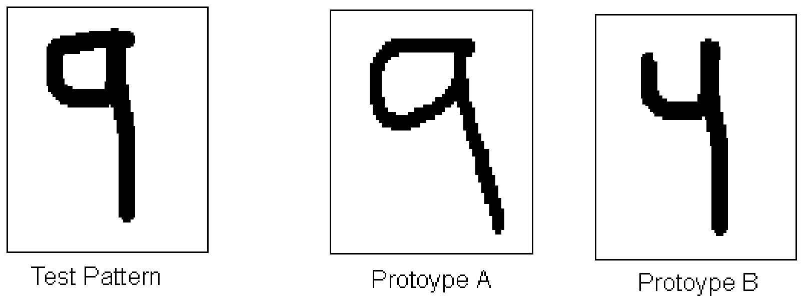


Figure 2.1: The test image of the digit ’9’ on the left has a closer Euclidean distance to prototype B, a digit 4, than prototype A, a 9. This is because Euclidean distance cannot account for the interaction between pixels or small variances in position and rotation.

could increase the number of training images to cover the entire domain of transformations (either by manually transforming the original training images or by gathering more data), but the combination of transformations will lead to an exponential increase in the required number of training images required.

Instead, we use tangent distance, which is a distance measure that linearly estimates all a priori transformations, to tractably emulate transforming the data. This will be discussed in section 2.2.3

2.2.2 Mahalanobis Distance

The Mahalanobis distance is a distance metric that can account for the correlations between different dimensions in the data. It is a powerful metric because it can normalize data that is skewed over multiple variables, and introduces scale invariance amongst the variables. This can be useful in real-world problems where we want to use variables with different units of measure in our calculations - for example, if we want to know the “difference” between two models of cars, we can use both the numbers of windows and the lengths of the cars. A Mahalanobis metric can normalize the units so the difference in number of windows isn’t weighted more or less than the difference in length. It can also be useful in image data, where some pixels may be correlated with each other more than others.

The intuition behind why the Mahalnobis distance can be better than the Euclidean distance is visualized in figure 2.2. The colors represent different labels of two dimensional data. On the left, the circle represents the contour of an Euclidean metric, and on the right a Mahalanobis metric. The Mahalanobis metric can separate the classes more effectively by following the general diagonal interaction of the two dimensions.

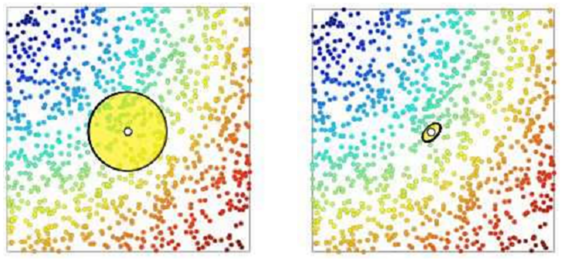


Figure 2.2: The Mahalanobis metric (right) more effectively follows the general diagonal interaction of the data than the Euclidean metric (left).

For two data points x and $y \in \mathbb{R}^d$, we define their Mahalanobis distance as:

$$D(x, y) = \sqrt{(x-y)^{\mathsf{T}} M (x-y)} \tag{2.3}$$

where M is the Mahalanobis matrix $\in \mathbb{R}^{dxd}$. The Mahalanobis distance reduces to the Euclidean distance if M is identity. M can be used to independently scale the dimensions if it is a diagonal matrix with scaled entries. Sometimes, it will be useful to decompose M as $M = L^{\mathsf{T}} L$ and write 2.3 as:

$$D(x, y) = \|L(x-y)\| = \sqrt{(x-y)^{\mathsf{T}} (L^{\mathsf{T}} L)(x-y)} \tag{2.4}$$

Given a dataset, we can use the distribution of the data to generate a Mahalanobis matrix that corrects for both scaled differences within the variables and the covariances between different variables. We do this by setting $M = \Sigma^{-1}$ where Σ is the covariance matrix of the data. Alternatively, it is possible to learn a Mahalanobis matrix to optimize some other function of the data, as we will see in Neighborhood Components Analysis (section 2.3.1) and Large Margin Nearest Neighbors Classification (section 2.3.2).

To understand the methods discussed in this thesis, it is helpful to remember a few properties of the Mahalanobis metric. To be a proper distance metric, $(x-y)^{\mathsf{T}}M(x-y)$ must be positive for any vectors x and y. Therefore, $a^{\mathsf{T}}Ma \geq 0$ for all a, which means M is positive semidefinite. The second property of the Mahalanobis metric we look at is a result of the symmetric constraint of distance functions :$(x-y)^{\mathsf{T}}M(x-y) = (y-x)^{\mathsf{T}}M(y-x)$. Simplifying this formula leads to $c^{\mathsf{T}}Md = d^{\mathsf{T}}Mc = (d^{\mathsf{T}}Mc)^{\mathsf{T}} = c^{\mathsf{T}}M^{\mathsf{T}}d$ where $c = (x-y)$ and $d = (y-x)$. Therefore, $M = M^{\mathsf{T}}$, which implies M is symmetric. Another important property is that the squared Mahalanobis distance is a linear function of the elements of M, which will be useful when learning the Mahalanobis metric.

2.2.3 Tangent Distance

As discussed in section 2.2.1, Euclidean distance is highly sensitive to small transformations that can affect many dimensions; translating an image just a few pixels can change a large percentage of the pixels in the image. This is not the expected behavior of a distance function that should return values that correspond to how "similar" or "dissimilar" two data points are. Tangent distance [Simard, LeCun, and Denker, 1993] allows us to determine what transformations the classification should be invariant to, and then build them into the function.

To understand tangent distance, it helps to first understand the theoretical motivation, starting with the idea of modeling possible transformations as a manifold. If we have N continuous, smooth transformations that can be applied to any D-dimensional data point, each data point exists on an (at most) N dimensional manifold that consists of all possible transformed points. For example, an image and the same image rotated 30 degrees both exist on the same 1-dimensional manifold of rotations in $\Re^D$ where D is the number of pixels in the image. This manifold can be pictured as a string that curves around high dimensional space (and eventually closes in on itself because rotation is a closed transformation). As we rotate the original image more and more, we move along the manifold.

Formally, a manifold is a subspace that is not necessarily Euclidean, but locally at any point resembles Euclidean space [Munkres, 2000]. For example, the manifold of rotations will curve around $\Re^D$, but it and its closest neighbors (the image rotated clockwise and counterclockwise) will form a 1-dimensional Euclidean subspace - they exist on the same line. If we have N possible transformations, the manifold will locally resemble an N dimensional hyperplane.

A brute force distance that takes into account our invariant transformations would, given two points and their respective manifolds, return the closest Euclidean distance between the two manifolds. Using the rotation example from above, this would be the equivalent of looking at every possible rotation of each image, and finding the pair of images from the two manifolds that have the least Euclidean distance. The digit '3' and another digit '3' that is slanted will return a small distance, as we would expect. Of course, this becomes intractable as the complexity of our transformations increase; the number of images we would need to generate is exponential in N.

We can drastically reduce this complexity by approximating the N-dimensional manifold by fitting an N-dimensional tangent plane to it, at the data point of interest. In doing so, we are assuming the local Euclidean structure of the manifold is approximately accurate as we transform the data point by a non-infinitesimal amount. Instead of finding the shortest Euclidean distance between the two manifolds, we find the shortest distance between the two tangent planes, which is called the tangent distance. The benefit of this is tractability - we will show a closed form solution to the tangent distance below.

If we have a set of N smooth, continuous, transformations, we can define a parametric equation, $s(x, \alpha)$ for the data points generated by transforming a data point x with parameter $\alpha \in \Re^N$. For example, α_1 may parametrize horizontal translation while α_2 parameterizes vertical translation, so if α_1, $\alpha_2 > 0$, the image $s(x, \alpha)$ will be image x translated up and right. The manifold of points created by this parameterization is:

$$S_x = \{y | \exists \alpha \text{ for which } y = s(x, \alpha)\} \tag{2.5}$$

We are interested in the tangent plane, P_x, to S_x at x. P_x is the first-order Taylor expansion of S_x when α is near 0:

$$P_x = s(x,0) + [\frac{\partial s(x,\alpha)}{\partial \alpha_1}, \ldots, \frac{\partial s(x,\alpha)}{\partial \alpha_N}]\alpha = x + T_x\alpha \qquad (2.6)$$

where $T_x = [\frac{\partial s(x,\alpha)}{\partial \alpha_1}, \ldots, \frac{\partial s(x,\alpha)}{\partial \alpha_N}]$ is called the tangent vector, and is the Jacobian matrix of the manifold with respect to α. Figure 2.3 shows this linear approximation with a one-dimensional manifold in $\Re^2$. Figure 2.4 shows a 256-dimensional example of image data, which shows that the tangent plane gives a good approximation to rotation when α is small.

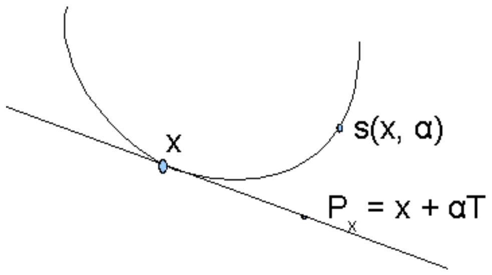


Figure 2.3: A 2-dimensional example of a manifold and its tangent plane (P_x). A point s is shown parameterized by α, with the corresponding approximation on P_x.

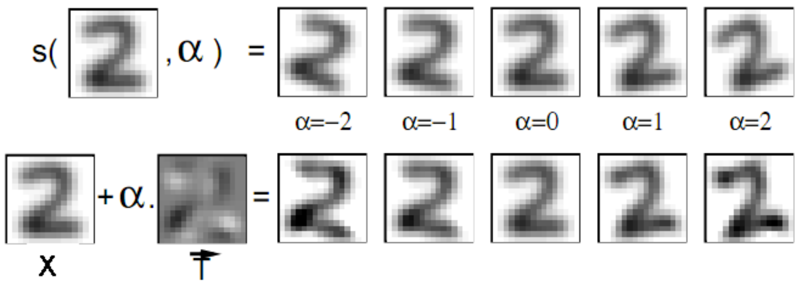


Figure 2.4: Applying a rotation transformation to the digit '2' (top) and its linear approximation (bottom). The approximation is accurate for small α, but becomes worse as α increases. [Simard et al., 1993]

Traditional tangent distance only seeks the shortest Euclidean distance between two tangent planes, P_x and P_y. We are also interested in the shortest (Euclidean) vector between the planes, which we call the shortest tangent vector. To calculate this vector, we find where the tangent planes are closest, and then return the vector between them:

$$\arg\min_{\alpha_x,\alpha_y} ||P_x(\alpha_x) - P_y(\alpha_y)|| \tag{2.7}$$

Finding α_x and α_y is a fairly straightforward linear least squares problem. We minimize 2.7 by setting its partial derivatives with respect to α_x and α_y to 0:

$$\frac{\partial ||P_x(\alpha_x) - P_y(\alpha_y)||}{\partial \alpha_x} = 2(P_x - P_y)^\intercal T_x = 0 \tag{2.8}$$

$$\frac{\partial ||P_x(\alpha_x) - P_y(\alpha_y)||}{\partial \alpha_x} = 2(P_y - P_x)^\intercal T_y = 0 \tag{2.9}$$

We can substitute P_x and P_y in 2.8 and 2.9 using 2.6:

$$T_y^\intercal(x - y - T_y\alpha_y + T_x\alpha_x) = 0 \tag{2.10}$$

$$T_x^\intercal(x - y - T_y\alpha_y + T_x\alpha_x) = 0 \tag{2.11}$$

This linear system of questions has the following solution [Simard et al., 1993]:

$$\alpha_y = \frac{((T_y^\intercal T_x)(T_x^\intercal T_y)^{-1}(T_y^\intercal) - (T_x^\intercal))(x - y)}{(T_y^\intercal T_x)(T_x^\intercal T_x)^{-1}(T_x^\intercal T_y) - (T_y^\intercal T_y)} \tag{2.12}$$

$$\alpha_x = \frac{((T_x^\intercal T_y)(T_y^\intercal T_y)^{-1}(T_x^\intercal) - (T_x^\intercal))(x - y)}{(T_x^\intercal T_x) - (T_x^\intercal T_y)(T_y^\intercal T_y)^{-1}(T_y^\intercal T_x)} \tag{2.13}$$

The tangent distance is $||P_x(\alpha_x) - P_y(\alpha_y)||$, but in our method we instead apply a Mahalanobis metric directly to the shortest tangent vector, $(P_x(\alpha_x) - P_y(\alpha_y))$.

An example of two images transformed using the tangent distance can be seen in figure 2.5. In this figure, two hand-written digit 1s (Image 1 and Image 2) are slight translations of each other - Image 2 is Image 1 translated left. The translation is large enough that only a few pixels overlap, leading to a high Euclidean distance. After transforming the images so they are as close as possible on their respective tangent planes, we get an approximate translation of the digits (second column) so they are closer to each other. The resulting L1 difference between the images, pictured in the bottom row, reflects the fact that tangent distance transforms the digits to be ”closer.” The Euclidean distance of the images is 3.8549 units, while the tangent distance is 2.6881 units

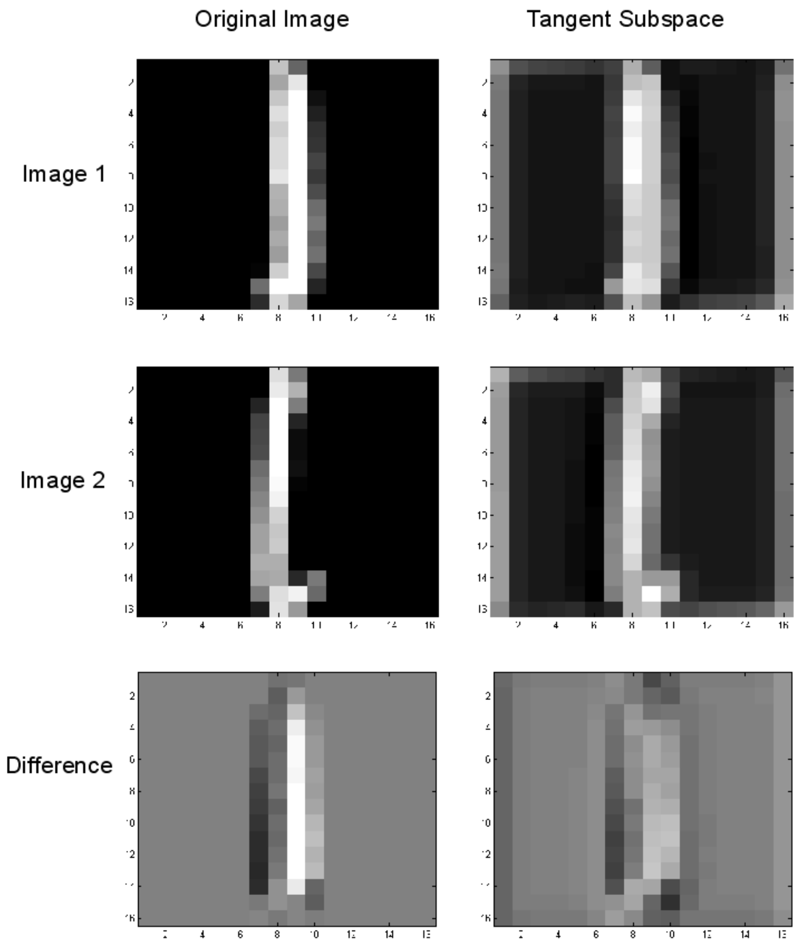


Figure 2.5: The shortest tangent vector (visualized in bottom right) has a smaller norm than the vector between the two points (bottom left) because the images lie close to the same tangent plane. (see text for detail.)

Note that the tangent distance is not a proper distance function, because $d(x, y) = 0$ when $x \neq y$ if x and y exist on the same tangent plane. This is fine because that is the expected behavior of the algorithm - if two data points are the same after an invariant transformation, we really do want their distance to be zero.

In practice, the tangent vectors T_x and T_y are found by transforming x and y by a small amount with each possible transformation, vectorizing the resulting points, and then placing them side-by-side into a matrix. Then, 2.12 and 2.13 are applied to all training images with the test image. The training tangent vectors, as well as $T_x^{\mathsf{T}} T_x$ and $(T_x^{\mathsf{T}} T_x)^{-1}$ can be saved in memory to save computation time.

In our experiments, we used the following six transformations: horizontal and vertical translation, scaling, rotation, and two hyperbolic transformations (see [Simard, LeCun, Denker, and Victorri, 1998] for more detail). We also experimented with line thickness and brightness transformations, but found they have no effect on performance.

2.3 Learning Mahalanobis Metrics for KNN

In KNN, a test point is classified by a vote of the k closest training points. Optimally, we want a test point to be close to at least a few training points with the same target value, and far from any training points with different target values. We seek a transformed metric space that naturally leads to this.

We will look at two different methods to optimize a Mahalanobis metric for nearest neighbors. The first, Neighborhood Components Analysis, optimizes an intuitive and direct objective function by gradient ascent in the space of Mahalanobis matrices. The second, Large Margin Nearest Neighbors, uses a less direct, but still intuitive objective function that allows the problem to be formulated as a convex optimization.

2.3.1 Neighborhood Components Analysis

Neighborhood Components Analysis (NCA) [Goldberger, Roweis, Hinton, and Salakhutdinov, 2005] is a simple algorithm to optimize a Mahalanobis metric for KNN classification. The objective function in NCA seeks to maximize the probability a test point would be classified correctly by 1-NN with the Mahalanobis metric. In the approach, the authors iterate through each training point, using the current Mahalanobis distance with all other points to determine a probability of correct classification. They then find the gradient of this probability with respect to the Mahalanobis matrix.

There are a couple of issues with calculating the gradient. When finding the probability of correct classification, the current training point must be excluded from the search for nearest neighbors; otherwise, every point would be classified correctly leading to a zero gradient. Another issue is that KNN is a discontinuous function that is not differentiable; to calculate the gradient of the objective function, the authors have to use a differentiable approximation to nearest neighbors. They assign a probability that each pair of points would be considered a neighbor, rather than assuming that only the closest point is a neighbor. They do this with the softmax function in the transformed Mahalanobis space

$$p_{ij} = \frac{\exp(-(x_i - x_j)M(x_i - x_j))}{\sum_{k \neq i} \exp(-(x_i - x_k)M(x_i - x_k))} \tag{2.14}$$

where p_{ij} is the probability that training points x_i and x_j are assigned as neighbors. The softmax function provides a smooth approximation to the step function.

If the target class of x_i is t_i, the set of all points in the same class as x_i is defined as $C_i = \{x_j | t_i = t_j\}$. The probability that x_i will be correctly classified can be denoted as:

$$p_i = \sum_{j \in C_i, x_j \neq x_i} p_{ij} \tag{2.15}$$

The objective function is the expected number of points correctly classified:

$$f(M) = \sum_i p_i \tag{2.16}$$

As mentioned in section 2.2.2, the optimization is over a matrix L instead of M, where $M = L^T L$. The authors differentiate f with respect to L to find the gradient rule (where $x_{ij} = x_i - x_j$:

$$\frac{\partial f}{\partial L} = -2L \sum_i \sum_{j \in C_i} p_{ij} (x_{ij} x_{ij}^T - \sum_k p_{ik} x_{ik} x_{ik}^T) \tag{2.17}$$

They reorder the equation to increase the calculation efficiency:

$$\frac{\partial f}{\partial L} = -2L \sum_i \left(p_i \sum_k p_{ik} x_{ik} x_{ik}^T - \sum_{j \in C_i} p_{ij} x_{ij} x_{ij}^T \right) \tag{2.18}$$

Any gradient optimizer can be used with this equation. In our tests, we used online gradient ascent, with a randomly initialized value of L.

2.3.2 Large Margin Nearest Neighbors

Large Margin Nearest Neighbors (LMNN) [Weinberger, Blitzer, and Saul, 2005], as in NCA, seeks a Mahalanobis metric to optimize for nearest neighbors. Unlike NCA, whose objective function is an optimization over the number of points that would be correctly classified, LMNN seeks to create locally favorable conditions for nearest neighbors.

There are a couple of intuitions to motivate the differences between the objective functions in NCA and LMNN. In NCA, every point is compared to every other point during the optimization. Realistically, in KNN, only training points that are close to a test point influence its classification; k points from the same class should be near, but any more than k points will not help classification; in fact, optimizing over more than k neighbors could potentially be harmful by transforming the metric too much. Therefore, in LMNN, each point is designated k “target neighbors” that are optimized over.

Another difference between NCA and LMNN is how they treat points with different labels. In NCA, every pair of points with different labels influences the error function. Again, this global optimization is unnecessary because distant points will not influence classification. In LMNN, the authors only optimize over nearby points with different target labels, which are referred to as "impostors". They define a "margin" around each point that impostor points should remain out of. The process of "pushing out" impostor points and pulling in target neighbors is visualized in figure 2.6.

As mentioned above, each data point has a set of k "target neighbors" that should be as close as possible to the point. The target neighbors must be provided to the algorithm; they are generally chosen as the k nearest neighbors using some fixed metric, such as Euclidean distance. As the Mahalanobis matrix is optimized, the identity of the target neighbors (which points are target neighbors) remains fixed. Each point has a margin that ideally are free from impostors. This margin is a radius around the point whose size is determined by the distance from its target neighbors. Therefore, as target neighbors get further away, the margin increases in size. The optimization only factors differently-labeled points that enter the margin, because these are the points that threaten to be classified as a nearest neighbor.

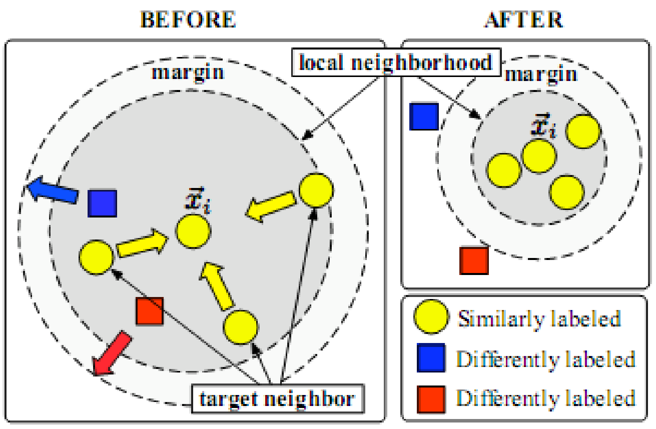


Figure 2.6: LMNN "Pushing Away" Impostors and "Pulling in" Target Neighbors [Weinberger et al., 2005]

LMNN's objective function, 2.19, has two competing terms based on the above intuitions. The first penalizes large distances from the target neighbors, and the second penalizes differently labeled points that enter the margin. While the first term tries to bring some points closer, the second tries to push others out.

$$\epsilon(L) = \sum_{ij} \eta_{ij} ||L(x_i - x_j)||^2 + c \sum_{ijl} \eta_{ij}(1 - T_{il})[1 + ||L(x_i - x_j)||^2 - ||L(x_i - x_l)||^2]_+ \tag{2.19}$$

Where $[z]_+ = max(z, 0)$ is the hinge loss. $\eta_{ij} \in 0, 1$ is a binary indicator that x_j is a target neighbor of x_i. $T_{ij} \in 0, 1$ is a binary indicator that x_i and x_j have the same labels: $t_i = t_j$. $c > 0$ is a parameter that weighs the cost of distant target neighbors with nearby impostors and can be found empirically (using cross-validation).

The first term in 2.19 simply sums the distances of all target neighbors, penalizing any that drift away. The second term penalizes only differently labeled points that enter the margin, equal to their distance from the margin.

2.19 cannot directly be optimized due to the discontinuous hinge function. Instead, the authors reformulate it as a semidefinite programming (SDP) problem - a convex optimization. SDPs are an optimization of symmetric semidefinite matrices over a linear cost function with linear constraints. As shown in section 2.2.2, the Mahalanobis metric is symmetric and positive semidefinite, and the squared Mahalanobis distance is a linear combination of the elements of the matrix. The first term of 2.19 is linear in the squared Mahalanobis distance, and is therefore a linear combination of the Mahalanobis matrix, M. The second term's hinge loss can be reformulated with slack variables to be linear in M. Therefore, we can reformulate the optimization to be over a linear combination of a semidefinite matrix with linear constraints, an SDP optimization.

Minimize:

$$\sum_{ij} \eta_{ij}(x_i - x_j)^T M(x_i - x_j) + c \sum_{ijl} \eta_{ij}(1 - T_{il})\xi_{ijl} \tag{2.20}$$

Subject to:

$$(x_i - x_l)^T M(x_i - x_l) - (x_i - x_j)^T M(x_i - x_j) \geq 1 - \xi_{ijl} \tag{2.21}$$

$$\xi_{ijl} \geq 0 \tag{2.22}$$

$$M \succeq 0 \tag{2.23}$$

where $M \succeq 0$ indicates that M is positive semidefinite, and ξ are slack variables. The slack variables mimic the hinge distance because they are set to 0, negating the second term in the minimization, unless an impostor enters the margin. When an impostor enters the margin, the slack variable equals the distance of the impostor from the margin, and is added to the error. This allows us to re-form the hinge loss as a linear function with linear constraints.

This optimization can be solved with any SDP solver, but the performance of most solvers scales badly as the number of constraints increase; in this optimization, the slack variables ξ_{ijl} each require an additional constraint. In our tests, we used a specialized solver provided by [Weinberger et al., 2005] that takes advantage of the fact that most $\xi = 0$ (because impostors are relatively rare.)

Chapter 3

Review of Relevant Literature

3.1 Distance Functions

As mentioned in sections 2.1 and 2.1.1, distance functions are critical to nearest neighbors classification and other non-parametric methods. Distance functions have been studied extensively in machine learning. An early example is [Michalski, Stepp, and Diday, 1981], which discusses the importance of using distance functions as a quantitative measure of similarity between objects. They use distance functions to group objects into clusters that are "similar" to each other. In doing so, they make the assumption that input objects that belong in the same cluster will be close to each other in input space using the chosen distance function. This is very similar to the assumption made in classification that input objects that are members of the same class will be close to each other in input space using the chosen distance function. In their clustering algorithm, the objects are clustered using a "cluster representation" in conjunction with a distance function. A cluster representation is a simple and general description of a cluster - for example, the centroid of a cluster can represent the entire cluster if it is assumed that each point belongs to the cluster with the closest centroid. A locally-optimal clustering is then found iteratively with two steps: the first, called the representation function, finds the optimal representation given the current cluster of each object. For example, if the cluster's centroid is the representation, each cluster's representation will be redefined as the centroid of the objects that are currently in that cluster. The second step, called the allocation function, is the inverse of the first step - given the new cluster representations, the objects are redistributed to the optimal clusters. In the cluster centroid example, moving the cluster centroids may have caused some objects to belong to new clusters. In the allocation step, these points are reallocated to their new clusters. By repeating these two steps, a locally optimal clustering of the points converges with the chosen distance function. One major difference from our approach is that they use a fixed distance function during the training, whereas we learn the distance function during training. This is due to the fact that our method allows the use of training labels, while their clustering is entirely unsupervised; we fix the representation and learn the distance function, whereas they fix the distance function and learn the representation. They discuss several distance functions, all of which are statistically derived. One such function, the "City Block" distance function, is simply a weighted sum of the difference of each dimension:

$$d(x,y) = \sum_{i=1}^{n} w_{(i)} |x_{(i)} - y_{(i)}| \tag{3.1}$$

where $x_{(i)}$ is the i^{th} element of x and $w(i)$ is a predefined weight on the i^{th} dimension. This distance function gets its name from a real-world example in the two dimensional case; if we want to find the shortest distance between two points in a city grid, we are forced to walk down one dimension, then the next (e.g. first take the east-west streets, then the north-south avenues), because it is usually impossible to walk through a block in a city. Another similar distance function mentioned in [Michalski et al., 1981] is the Chebyshev distance function:

$$d(x,y) = \max_{i} |x_{(i)} - y_{(i)}| \tag{3.2}$$

The Chebyshev function is mostly useful when the dimensions are normalized, because it only uses the dimension with the maximum difference between the two points. This function can be useful if we are only interested in using a single attribute to compare two objects, but do not know which a priori. For example, if we are comparing models of cars, we may be interested in both the price and the quality of the car. If the quality of two models of cars is very different but the price is similar, we may still consider the two models to be very different, and vice versa. Of course, we would have to normalize the two attributes to ensure that one does not dwarf the other. Many other distance functions are mentioned, including the "quadratic" distance function, which is the same as the Mahalanobis distance function discussed in section 2.2.2.

3.1.1 Nominal Attributes

[Stanfill and Waltz, 1986] discuss distance metrics that take data points with only nominal (categorical, non-continuous) attributes, as opposed to continuous attributes. A nominal attribute is discrete and has no intuitive ordering. For example, if our data are cars, possible nominal attributes are make, model, and color. Because there is no intuitive order or magnitude to these attributes, traditional distance metrics (which depend on the sum and difference of values) cannot be used. The authors begin by introducing the "overlap metric", which is simply the number of attributes that are different between two data points:

$$d(x,y) = \sum_{i=1}^{n} (x_{(i)} \neq y_{(i)}) \tag{3.3}$$

For example, if two cars are of different make, model, and color, their overlap dissimilarity would be 3. If they were the same color, it would be 2. Although simple, this is a poor metric because it assigns an equal weight to all attributes and classes. The authors take a statistical approach to the problem with the Value Difference Metric (VDM):

$$d(x,y) = \sum_{i=1}^{n} \sum_{k=1}^{C} \left| \frac{N_{i,x,k}}{N_{i,x}} - \frac{N_{i,y,k}}{N_{i,y}} \right| \tag{3.4}$$

where:

- i enumerates over all attributes (dimensions) of the data
- k enumerates over the set of all possible output classes, C

- $N_{i,x}$ is the number of points in the training dataset that have value x from attribute i
- $N_{i,x,k}$ is the number of points in the training dataset that have value x from attribute i and output class k

VDM statistically determines the similarity of two objects' based on the proportion of the number of times their particular attributes are in the same class. For example, our goal may be to classify whether a car is a sports car or family car based on its color; the two possible classes are sports and family, and its attribute is color. If we are comparing a red car with a beige car, we will see a big distance if most red cars in our training set are sports cars, and most beige cars in our training set are family cars. On the other hand, a beige car and pink car will have a small distance if the training set reflects that both are seen in family cars more than sports cars. VDM and the overlap metric expect nominal attributes, and generally fail for continuous attributes. This makes sense - in many continuous attributes, there are very few overlaps. This is especially true for non-integral values - for example, car mileage (what are the chances that two cars have *exactly* the same mileage?)

[Wilson and Martinez, 1997] explore extending these distance functions for situations when continuous and nominal attributes are mixed. We may be interested in both the color of a car and its mileage. They extend the overlap metric with the Heterogeneous Euclidean-Overlap Metric (HEOM). If an attribute is nominal, its contribution to the distance is the same as in equation 3.3, but if it is continuous its contribution to the distance is defined by the normdiff function:

$$d(x,y) = \sum_{i=1}^{n} \begin{cases} \text{overlap}(x,y) \text{ if attribute i is nominal} \\ \text{normdiff}(x,y)^2 \text{ if attribute i is continuous} \end{cases} \tag{3.5}$$

where

$$\text{overlap}(x,y) = \begin{cases} 0 \text{ if } x = y \\ 1 \text{ if } x \neq y \end{cases} \tag{3.6}$$

and

$$\text{normdiff}(x, y) = \frac{|x - y|}{range_i} \tag{3.7}$$

In equation 3.7, $range_i$ denotes the ranges of values of the attribute i:

$$range_i = max_i - min_i \tag{3.8}$$

Note that equations 3.3 and 3.6 are the same; HEOM only differs from the overlap metric when an attribute is continuous. To mix continuous and nominal attributes, we must normalize the continuous attributes so they do not have more or less weight than the nominal attributes. In the overlap metric, no attribute could contribute more than 1 to the distance, so we normalize the difference in eq 3.7 with the range of that attribute. This way, an attribute with typically high values (e.g. mileage on a car) will have the same min and max values as one with small values (e.g. number of windows). HEOM suffers from the same deficiencies as the overlap metric.

Wilson and Martinez discuss several possible ways to extend VDM to the continuous case to overcome these issues. One - discretization, involves splitting each continuous attribute into discrete bins so that each bin represents a nominal value. After this discretization, VDM can be run on the data. This approach cannot take advantage of the ordering of the bins - the first bin and the last bin are likely more ”different” than neighboring bins. Another problem with discretization is that values on the two extremes of each bin are considered just as similar as two identical values. By treating continuous variables as nominal, we lose a lot of information. Another alternative to VDM is the heterogeneous value difference metric (HVDM). HVDM, like HEOM, normalizes the continuous attributes to fit into the nominal framework; we first normalize the continuous attributes, then treat them as nominal in the original algorithm. HVDM is calculated as:

$$d(x, y) = \sum_{i=1}^{n} \begin{cases} \text{normalized\_vdm}(x, y)^2 \text{ if attribute i is nominal} \\ \text{normalized\_diff}(x, y)^2 \text{ if attribute i is continuous} \end{cases} \tag{3.9}$$

Normalizing these values requires some careful thought, because nominal attribute distances are calculated differently from continuous attribute distances. Continuous attribute distances are computed by subtracting the two input values, whereas nominal attributes distances are sums of probabilities over the classes. This is in contrast to the overlap metric and HEOM, where both the continuous and nominal attribute distances were computed by subtraction of values. Therefore, the results of normalized_diff need to be distributed similarly to normalized_vdm. This means its range should be close to 0 to 1. We cannot simply divide $|x - y|$ by $range_i$ as in equation 3.7, because this can cause the distribution of normalized_diff to be squeezed mostly into a small range. Instead, we assume that normalized_diff is normally distributed, and that we want most of its values to fall into the range 0 to 1; the authors chose:

$$\text{normalized\_diff}(x, y) = \frac{|x - y|}{4\sigma_i} \tag{3.10}$$

where σ_i is the standard deviation of the values in attribute i. In a normal distribution, 95% of values fall within two standard deviations of the mean - therefore, normalizing the distribution by $4\sigma_i$ should bring most values within the range 0 to 1 (of course $|x-y|$ cannot be exactly normally distributed because it is always a positive value). The authors studied several possibilities for normalized_vdm, which can be as simple as the term from eq 3.4:

$$\text{normalized\_vdm\_1}(x, y) = \sum_{k=1}^{C} \left| \frac{N_{i,x,k}}{N_{i,x}} - \frac{N_{i,y,k}}{N_{i,y}} \right| \tag{3.11}$$

A few other possibilities:

$$\text{normalized\_vdm\_2}(x, y) = \sqrt{\sum_{k=1}^{C} \left| \frac{N_{i,x,k}}{N_{i,x}} - \frac{N_{i,y,k}}{N_{i,y}} \right|^2} \tag{3.12}$$

$$\text{normalized\_vdm\_3}(x, y) = \sqrt{|C| * \sum_{k=1}^{C} \left| \frac{N_{i,x,k}}{N_{i,x}} - \frac{N_{i,y,k}}{N_{i,y}} \right|^2} \tag{3.13}$$

where $|C|$ is the number of classes. Over 15 different datasets, the authors experimentally found normalized_vdm_2 to generalize the best, and normalized_vdm_1 the least. The authors also compared HVDM with Euclidean and HOEM by implementing a nearest neighbors classifier with each, and running them on a set of 15 machine learning datasets. They found HVDM to be superior to both Euclidean and HOEM. A few conclusions can be drawn from their experiments concerning classification using nominal and continuous data: treating nominal values as continuous (Euclidean) is a bad idea, as is treating continuous values as nominal, which requires discretization. Additionally, a statistical approach (VDM and HVDM) is superior to counting the number of different attributes (overlap metric and HOEM).

In our research, we only looked at continuous attributes - pixel values - for classification. Modifying our method to deal with nominal values would be difficult because there is no intuitive tangent subspace for nominal values. Even if we fixed the values of the nominal attributes in the first tangent step, learning a Mahalanobis metric would be difficult on nominal variables. One possible solution could be to use our approach to find a distance over all the continuous attributes, and then combine this term with all nominal attributes in a weighted HVDM step; in this approach, our continuous distance would replace the $|x - y|$ term in equation 3.10.

3.2 Efficiency of Nearest Neighbors With Tangent Distance

[Hastie and Simard, 1995] point out that nearest neighbors is expensive because a test point must be compared to every training point; this is especially costly with complex distance functions such as the tangent distance, as we discuss in section 4.3.1. They explore using clustering to improve the efficiency of nearest neighbors when using tangent distance.

They discuss two methods. The first, called the tangent centroid, clusters a group of training points (one group for each label, for example) into a single point that seeks to minimize loss in classification accuracy. Specifically, the tangent centroid, M_T, of a set of points x, minimizes the average squared tangent distance between the centroid and the points:

$$M_T = \arg\min_M \sum_{i=1}^{N} d(x_i, M)^2 \tag{3.14}$$

where d is the tangent distance. The cost function to be minimized is

$$C(M) = \sum_{i=1}^{N} \min_{\alpha_i, \gamma_i} \|M + T(M)\gamma_i - x_i - T_i\alpha_i\|^2 \tag{3.15}$$

where T_i is the tangent subspace of x_i, $T(M)$ is a function that returns the tangent subspace of M, α_i is the tangent plane parameterization of x_i and γ_i is the tangent plane parameterization of M when computing the tangent distance to x_i (see section 2.2.3 for more detail). We optimize 3.15 by initializing M as the Euclidean centroid of x, then iteratively performing the following steps until D converges:

1. Find the α_i and γ_i for all x_i that minimizes $\|M + T(M)\gamma_i - x_i - T_i\alpha_i\|$
2. Set $M \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_i + T_i\alpha_i - T(M)\gamma_i)$
3. Compute $D = \sum_{i=1}^{N} d(x_i, M)$ where d is the tangent distance

The second method that Hastie and Simard discuss, called the tangent subspace, replaces the a priori tangent subspace computation with a statistical computation, which eliminates the need to compute the tangent subspace in each iteration of the optimization. Instead of choosing m a priori transformations, we now choose a fixed number (which we also call m) of transformations that will parametrically and statistically be derived during the optimization. We call this parameterized tangent subspace V. With this modification, we can simultaneously compute both the cluster centroids and the m transformations during the optimization. One benefit is that we are no longer limited to the seven or eight a priori transformations we used in section 2.2.3 - m is now a free parameter. The downside is that the power of hand picking the a priori transformation is no longer there - we are now limited to what we can statistically derive. In each iteration of the tangent subspace algorithm, we recalculate the tangent subspace to be the closest m-dimensional affine tangent plane to the cluster points (minus the centroid). This m-dimensional plane can be derived using the SVD: $SVD(x - M) = UDW^T$; it is simply the first m columns of W. To compute the cluster centroid and tangent subspace, we first initialize the centroid M to the Euclidean centroid of x and set V as the first m right singular vectors of x. We then perform the following steps until the squared sum of the matrix D (from the SVD in step 3) converges:

1. Find the α_i and γ_i for all x_i that minimizes $\|M + V\gamma - x_i - V\alpha_i\|$
2. Set $M \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_i + V\alpha_i)$
3. Compute the SVD of $x_i + V\alpha_i - M$. Set V to the first m right singular values of the SVD.

A test point's distance is compared to each centroid using its corresponding tangent subspace to find the nearest cluster, which is much quicker than comparing the distance to each training point.

Hastie and Simard also mention experiments with a similar method to ours: computing the Mahalanobis distance of each point to the tangent clusters. They found this hurt their performance. Their experiment differs from our method in a few important ways - first, they are computing the traditional Mahalanobis distance instead of an Mahalanobis distance on the shortest vector between the tangent planes. Also, they did not attempt to learn the Mahalanobis metric.

3.3 Combining A Priori and Statistical Information for Classification

[Fraser, Hengartner, Vixie, and Wohlberg, 2003b] integrate invariance to known a priori transformations with a class-based Mahalanobis classifier. This is similar to our approach because its goal is to classify test points by combining statistical information on the data with a priori transformations that the data should be invariant to. Our approach is fundamentally different, however: we learn a Mahalanobis metric between the test point's and training points' first order estimates of the invariant manifold, whereas they modify the inverse-covariance Mahalanobis metric using a second-order estimate of the invariant manifold. They begin with the classic Mahalanobis distance classifier:

$$t(y) = \arg\min_c (y - \mu_c)^T \Sigma_w^{-1} (y - \mu_c) \tag{3.16}$$

where y is the point to be classified, t is a function that outputs a class prediction, μ_k is the centroid of class c, and Σ_w is the covariance of all same-class points. In this classic Mahalanobis classifier, a test point is classified by comparing its Mahalanobis distance to all class centroids. Their strategy is to augment the Mahalanobis classifier with the second order approximation to the manifold of the invariant transformations on the class centroids. The second order Taylor expansion of the manifold s at a point Y is:

$$s(Y, \alpha) = s(Y, 0) + V\alpha + \alpha^T H \alpha + R \tag{3.17}$$

where R is the remainder, α parameterizes the invariant transformation (as in section 2.2.3) and

$$(V_c)_i = \frac{\partial s(Y_c, \alpha)}{\partial \alpha}\bigg|_{\alpha=0} \tag{3.18}$$

$$(H_c)_{i,j} = \frac{\partial^2 s(Y_c, \alpha)}{\partial^2 \alpha_i \alpha_j}\bigg|_{\alpha=0} \tag{3.19}$$

To augment 3.16 with the second-order approximation of the invariant transformations at μ_c (3.17), they replace Σ_w with Σ_c:

$$\Sigma_c = \Sigma_w + \beta V_c C_{\alpha,k} V_k^T \tag{3.20}$$

where $C_{\alpha,k}$ is a $m \times m$ positive semi-definite (PSD) matrix and β is a weighting parameter. When Σ_c^{-1} is used in 3.16 as a metric, $V_c C_{\alpha,c} V_c^T$ represents exploration of the tangent subspace spanned by V_c, with the exploration controlled by $C_{\alpha,c}$. They model α_c, the transformation parameter over the tangent subspace around μ_c, as a Gaussian distribution. $C_{\alpha,c}$ is the model of the covariance of α_c - as it spreads out more, α_c it deviates from the class mean, μ_c, more. While this encourages exploration of the tangent subspace, it causes the second order Taylor series 3.17 to deviate from the mean, and therefore approximate the true manifold worse. Therefore, $C_{\alpha,c}$ should balance the benefit of exploration with the disadvantage of decreased accuracy. Fraser discusses an optimization of $C_{\alpha,c}$ that the authors developed, which will be summarized here; for a more in-depth proof and further explanation of the procedure, see [Fraser et al., 2003b] and [Fraser, Hengartner, Vixie, and Wohlberg, 2003a]. For the remainder of this section, a single class is considered, so the subscript c is dropped. First, the authors take the eigen-decomposition of the within-class covariance matrix Σ_w to quantify the significance of the components (which is an estimate of cost of excursions in the direction of that component):

$$\Sigma_w = \sum_d e_d \lambda_d e_d^T \tag{3.21}$$

for each component d. They then break the $m \times n \times m$ tensor H into components:

$$H_d \equiv e_d^T H \tag{3.22}$$

Define:

$$\bar{H} \equiv \sum_d \sqrt{H_d^T H_d} \times |\lambda_d|^{-\frac{1}{2}} \tag{3.23}$$

Define the norm:

$$|\alpha|_{\bar{H}} \equiv \sqrt{\alpha^T \bar{H} \alpha} \tag{3.24}$$

The optimization is:
Maximize determinant $|C_\alpha|$
Subject to:

$$\mathbb{E}|\alpha|_{\bar{H}}^2 \leq \gamma \tag{3.25}$$

where γ is a constant. The solution to the optimization is in [Fraser et al., 2003b]:

$$C_\alpha = \beta(\bar{H})^{-1} \tag{3.26}$$

where β is a function of γ and balances the competing goals. As in our approach, [Fraser et al., 2003b] combines statistical information from the training dataset with an estimate of the manifold of invariant transformations to realize gains from both.

Our approaches are inherently different, however; while we optimize over the statistics of the data *after* they are transformed in the subspace of the first-order estimate of the manifold, they optimize a statistical transformation using both the statistics of the training data and the second-order estimate of the manifold simultaneously. Their approach depends on aggregating statistics over classes (or clusters) to combine with the second order estimate of the manifold. Therefore, they are constrained to the cluster-level Mahalanobis distance classifier. Our method, in contrast, optimizes a Mahalanobis metric over all points and classes to create a finer-grained nearest neighbors approach to classification. This is a benefit when large training sets that effectively represent the input domain are available (as in digit recognition).

3.4 Combining Support Vector Machines, Nearest Neighbors, and Tangent Distance

[Zhang, Berg, Maire, and Malik, 2006] use support vector machines (SVMs) for nearest neighbors classification. Instead of transforming the distance metric and then classifying over the transformed space (as in our approach), they seek to efficiently classify in one step. They do this by using a multi-class SVM to classify a test point using only its nearest neighbors. There are a few benefits to this approach: like Large Margin Nearest Neighbors [Weinberger, Blitzer, and Saul, 2005], the tractability of the problem is greatly increased by learning locally - because SVM is only applied to the nearest neighbors of a test point, the classification is limited to only a few classes and points. The authors argue that this leads to a more natural and better-behaving classification function than traditional multi-class SVM. Another benefit is that they can build a distance function into the kernel of the SVM using the kernel trick:

$$K(x, y) = \frac{1}{2}(< x, x > + < y, y > - < x-y, y-x >) = \frac{1}{2}(d(x, 0)+d(y, 0)-d(x, y)) \tag{3.27}$$

where d is distance function. The location of the origin does not affect SVM. The algorithm the authors develop, called SVM-KNN, has the following steps:

1. Find the k closest neighbors to the test point using any distance function

2. Compute the pairwise distance matrix of the test point and its k neighbors
3. Use the kernel trick to convert the pairwise matrix into a kernel matrix (eq. 3.27)
4. Apply multi-class SVM to the kernel matrix and label the test point with the resulting classifier

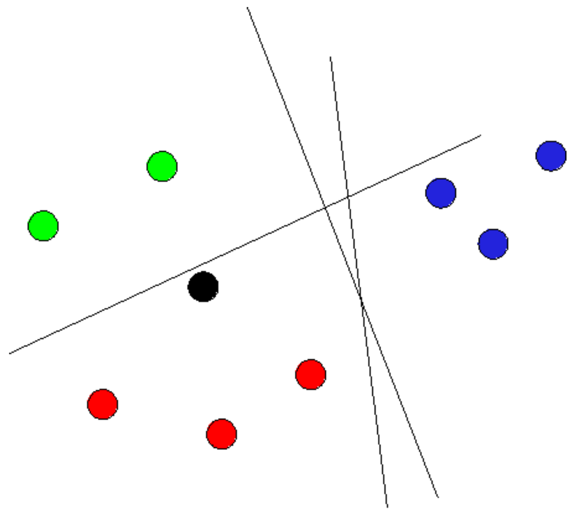


Figure 3.1: A visualization of SVM-KNN. In this example, the black test point will be classified in the red class. See text for more detail.

The final step involves choosing a multi-class SVM to classify the point. The authors analyzed several different methods, and chose DAGSVM [Platt, Cristianini, and Shawe-Taylor, 2000]. DAGSVM uses a directed acyclic graph (DAG) to combine the results of the 1-1 SVMs (SVM between each pair of classes) on the test point to determine the classification. The DAG is essentially a decision tree that attempts to split the input space into regions by the 1-1 SVM linear separators. For example, if we are trying to classify a test point between three classes (1, 2, and 3), we first eliminate class 1 or 2 from the possibilities by using the class 1-2 SVM (the linear separator between the points in class 1 and class 2). If class 2 is eliminated, we repeat this step using the class 1-3 SVM. If class 1 is eliminated, we classify the test point as a member of class 3. In [Zhang et al., 2006], DAGSVM is shown to work just as well as other methods, but is much quicker. SVM-KNN can be visualized in figure 3.1. In this figure, the test point (black point) is compared to its k (eight) closest neighbors. The multi-class SVM classifies the test point using a combination of the three different 1-1 separators. The 1-1 separators are generated from the kernelized distance matrix between the test point and its neighbors. In this example, the point will be classified in the red class.

The authors frame SVM-KNN as a compromise between KNN and SVM; when k is small, the classifier is very similar to nearest neighbors, because the SVM on a small number of points will usually heavily favor the most frequent class. On the other extreme, when $k = n$, SVM-KNN reduces to traditional multi-class SVM. The authors achieved results similar to K-NN with SVM-KNN (see table 4.3), but improved their accuracy considerably when they replaced the Euclidean distance with tangent distance. Using tangent distance with SVM introduces some issues, because the tangent distance is not a proper distance function (see section 2.2.3). A distance matrix produced using tangent distances may not produce a positive semi-definite kernel matrix because tangent distances can break the triangle equality. To get around this, the authors determine if the kernel matrix is not positive-definite by checking if the smallest eigenvalue of the kernel matrix is negative. If it is, they subtract this value from the diagonal entries of the kernel matrix (or add its absolute value) [Pekalska, Paclk, and Duin"]. They are effectively increasing the "self-similarity" of all points (which will not affect the similarity measure between points) while ensuring that the kernel matrix is positive-definite. SVM-KNN, similar our approach, depends on local methods to classify test points. Instead of transforming the space and then applying a classifier to the nearest neighbors, they apply a classifier directly to the nearest neighbors.

As in our approach, they first compute a tangent subspace. While we apply a Mahalanobis metric to the tangent subspace, they use the tangent distances to produce a kernel matrix for SVM. Both methods find improvement in nearest neighbors by using tangent distances to integrate a priori information into the classifier.

3.5 Generalizing LMNN

[Sriperumbudur and Lanckriet] generalize large margin nearest neighbors to use arbitrary metric spaces. They do this by finding a non-linear function that embeds the points from an arbitrary metric space into an Euclidean space, such that the leave one out (LOO) error of Euclidean nearest neighbors is minimized. Like LMNN, the problem is reformulated to be a convex semi-definite programming problem. The benefit to this approach is the transformation is not constrained to be linear.

Chapter 4

Approach

As we have seen, there are various ways to improve the Euclidean distance if we have some knowledge of our problem domain. Given a priori knowledge of possible transformations that our classification should be invariant to, tangent distance can be used to transform similar data so they are closer. The similarity of the data may not be evident in the statistics of the data, which is why using a priori knowledge can help the classification considerably.

In the absence of a priori knowledge, we can look at the statistics of the data to learn an optimized Mahalanobis metric for our problem. These learning algorithms can find patterns in the data that humans may not obviously see. They are automatic, in the sense that a human is not required to instill his own knowledge into the model.

These methods attack the deficiencies of Euclidean distance from different perspectives. Tangent distance compensates for sensitivity to small transformations using human knowledge, while a Mahalanobis metric automatically compensates for scaling and correlations in the dimensions of the data. Our method takes advantage of the fundamental differences in these two approaches by combining them. The key insight is how we combine them.

4.1 Combining Tangent Distance with a Mahalanobis Metric

In chapter 3, we discuss previous research that explores combining tangent distances with Mahalanobis metrics, including [Macherey et al., 2001] and [Fraser et al., 2003b]. Unlike previous methods, however, we combine them in a unique two-step process to take advantage of their strengths. Tangent distance brings pairs of similar points closer to each other, and then finds the Euclidean distance between the transformed points. Its main strength is the process of bringing points that are similar (after possible a priori transformations) closer. Mahalanobis metrics, on the other hand, simply act on a vector between two points; how it acts on that vector can be optimized for a particular model given a training dataset. This possible optimization is its main strength.

In our method, we first use tangent distance to bring similar points closer; we find the shortest tangent vector, $P_x(\alpha_x) - P_y(\alpha_y)$, as outlined in equations 2.12 and 2.13. We then apply a trained Mahalanobis metric to this vector (for example, one trained with NCA or LMNN, as discussed in sections 2.3.1 and 2.3.2. When training the Mahalanobis matrix, we optimize over the shortest tangent vector instead of the original pairs of points.

By combining tangent distance with learned Mahalanobis metrics, we seek to combine the best of both. We can intuitively see that they work well together: using the minimum tangent vectors should make the learning of the Mahalanobis matrix easier because there should be less variation between pairs of points with the same label.

Given a test point, y, and a set of n test data points, $\{x_1, x_2, \ldots, x_n\}$, we can classify y using 1-NN with the following steps:

1. For each x_i, find the shortest tangent vector between y using eqs. 2.12 and 2.13

2. Apply the Mahalanobis metric to each shortest tangent vector using eq. 2.3

3. Return the label t_i of the x_i with the shortest Mahalanobis distance

This process can take a considerable amount of time, due to the need to find the n shortest tangent vectors between y and the training points. An idea from [Simard et al., 1993] that we implemented in our classification algorithm is to only consider the 100 training points with the shortest Euclidean distances to y - all other points are so far away, it is unlikely that our method would transform them to be the nearest neighbor. This is considerably faster because finding the Euclidean distance to all training points is much faster than computing their tangent distances followed by a Mahalanobis transformation. In our tests we found this optimization rarely affected classification accuracy. See section 4.3.1 for more discussion on performance.

Obviously, the training has to be slightly modified for this method. Finding the shortest vector between tangent planes as described in section 2.2.1 remains the same. Instead of training the Mahalanobis metric on the vectors between the pairwise points, we train it on the shortest tangent vector. Therefore, the Mahalanobis metric is learned specifically for this process. It should also be possible to apply a Mahalanobis metric learned directly on the data (without a tangent transformation), but we found this always leads to lower accuracy. This may be due to the fact that the within-class variance of points and point-to-point L2 distance is lowered after transforming the points with the shortest tangent vector algorithm. Therefore, the Mahalanobis metric may be training on data that is further away and more ”different” than it will experience during testing.

4.2 Results

We tested our method on the United States Postal Service (USPS) database, which contains 9298 16×16 pixel gray-scale images of handwritten digits, with corresponding labels. The digits are evenly distributed from 0 to 9. The dataset is traditionally split into 7291 training and 2007 test images.

We classified the images using 1-NN using different distance functions. For comparison, we trained the Mahalanobis metric using both the shortest tangent vector and the point-to-point vector $(x_1 - x_2)$. We trained the Mahalanobis metric with LMNN

Table 4.1: Results - % Error on USPS Test Dataset, k=1

Mahalanobis Training Algorithm	Point-to-Point Vector	Shortest Tangent Vector
Identity	5.63	3.44
LMNN	5.58	3.09

Table 4.2: Results - % Error on USPS Train Dataset, k=1

Mahalanobis Training Algorithm	Point-to-Point Vector	Shortest Tangent Vector
Identity	2.83	1.33
LMNN	2.71	1.23

and compared this approach with the Euclidean distance. The identity Mahalanobis metric applied to the shortest tangent vector is the same as the tangent distance. The identity Mahalanobis metric applied to the point-to-point vector is the traditional Euclidean distance. We experimented with NCA, but our implementation did not scale well.

In all experiments, we initialized the matrix with the Identity matrix. This makes the comparison with the Euclidean distance and tangent distance fair.

Results can be seen in tables 4.1 and 4.2. Some results of other methods on the same dataset are shown in table 4.3.

Table 4.3: Results - % Error of Other Approaches on USPS Test Dataset

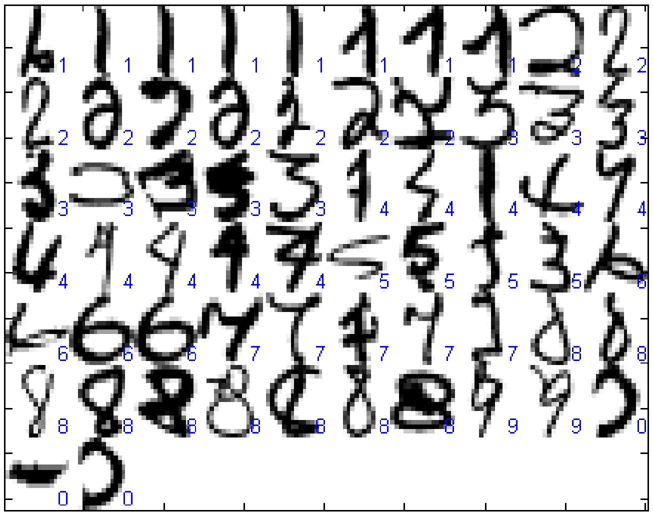
Approach	Error
Human	2.50 [Simard et al., 1993]
SVM-KNN (k=10) (On Euclidean Distances)	4.29 [Zhang et al., 2006]
SVM-KNN (k=8) (On Tangent Distances)	2.59 [Zhang et al., 2006]
Convolutional Neural Network (LeNet1)	4.2 [Simard et al., 1998]
Relevance Vector Machine	5.1 [Tipping, 2000]

4.3 Discussion

Our method (LMNN on the shortest tangent vector) performs well compared to other methods, including convolutional neural networks and relevance vector machines. SVM-KNN with tangent distance performed better than LMNN with tangent distance, however. While the tangent distance performs well by itself, LMNN offers a slight improvement. Given the tough nature of the problem (humans are shown to misclassify 2.5% of the test set [Simard et al., 1993], we consider a 0.3% reduction in error promising - especially because the additional cost of running the Mahalanobis step is small compared to computing the shortest tangent vector. Figure 4.1 shows all test images from LMNN on the shortest tangent vectors that were misclassified. Many of these test points would be difficult to classify for a human.

A possible problem with applying any Mahalanobis metric for classification is an averaging effect over the dimensions (pixels) between the different categories (digits). While pixel A and pixel B may be positively correlated between examples of the digit 0, that correlation may be negative between examples of the digit 1. The Mahalanobis metric, therefore, would have to compromise over this pair of pixels. This may be a general problem when learning a single transformation to separate multiple classes - the transformation has to be sufficiently powerful to distinguish each class from all others. Nevertheless, we found that it is possible to realize some performance gain, albeit small, by using Mahalanobis metrics. Note that tangent distance would not have the same problem because every digit should be invariant to the same transformations, so the shortest tangent vector really does offer separation from a class and all others.

A consideration when training any statistical algorithm on top of the shortest tangent vectors is that the shortest tangent vectors do not exist in a true metric space. For example, the shortest tangent vectors between three points do not have to follow the triangle inequality, unlike the point-to-point vectors. If an algorithm makes the assumption that the vectors provided to it must follow certain laws such as the triangle inequality, its application to the shortest tangent vector may be undefined and actually



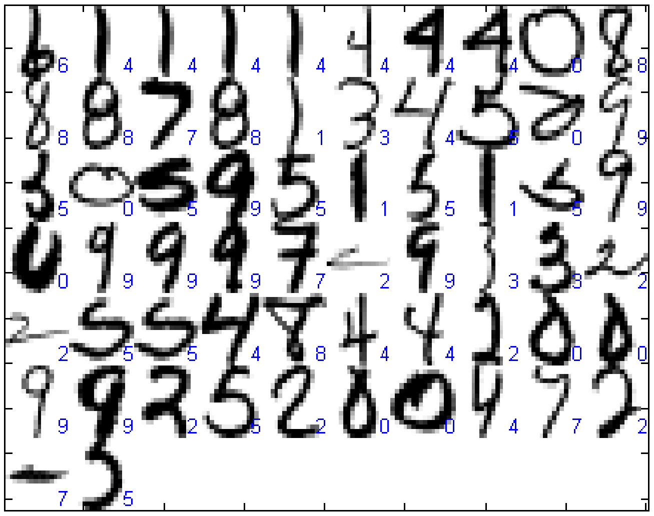


Figure 4.1: The top set of digits are incorrectly-classified test images by LMNN on the shortest tangent vectors. The bottom set are the corresponding nearest-image predictions from the training set. The blue text are the actual labels of the images.

lower accuracy.

4.3.1 Performance

Performance issues arise with our method during the tangent step: finding the shortest tangent vector is slow. Each test image must be transformed by six transformations, and then equations 2.8 and 2.9 must be run on the test images. Even after we only considered the 100 closest (in Euclidean space) training points (see section 4.1), it took about ten minutes to classify 2007 test images with 7291 training images on a 3 GHz CPU with 1.5 GB of RAM using unoptimized Matlab code. The Mahalanobis step performs quickly, with only two vector transformations required per squared distance calculation.

Training was considerably more time-consuming. When learning the Mahalanobis matrix, the shortest tangent vector for each pair of training images is required. Storing 7291^2 6x256 element matrices is prohibitive, so these values had to be constantly recalculated. If enough iterations of the learning algorithm are required, this may make learning the Mahalanobis matrix impossible.

4.4 Extension of Our Method to Regression

We would like to explore more ways of learning metrics on top of the shortest tangent vectors, including other Mahalanobis-based methods. Any learning algorithm that works on only the vectors between the data could have useful potential results on the shortest tangent vectors.

A possible extension to our method would be using it for regression instead of classification. In classification, we seek to assign a label $t \in C$ to a test point x; in regression, we assign a real number $t \in \mathbb{R}$ to the point. For example, if our data are different cars, we may be interested in assigning a prediction of the resale value of a car. This is a regression task, because the possible assignments are real, continuous numbers instead of nominal categories. In classification tasks we expect training data with class labels - in regression, we expect training data with continuous output values.

A possible way to extend our method would be to learn a Mahalanobis metric for regression. Weinberger and Tesauro (2007) developed a method to do so called Metric Learning Kernel Regression (MLKR) [Weinberger and Tesauro, 2007]. MLKR is very similar to NCA; unlike NCA and LMNN, MLKR uses kernel regression instead of nearest neighbors to assign values to test points. Kernel regression is a non-parametric regression algorithm that assigns a value to a test point based on a weighted average of its neighbors:

$$c(x) = \frac{\sum_i y_i K(x, x_i)}{\sum_i K(x, x_i)} \tag{4.1}$$

where c is the kernel regression function that outputs a real value, x is the test point to be classified, x_i are the training points, y_i are their values, and $K(a, b) > 0$ is the "kernel function". Kernel regression is visualized in figure 4.2. In this figure, if we assume the kernel function weights smaller distances more, the training point represented by the small blue circle will have a greater weight than the other training points because it is closer to the red test point. The kernel function is a function that takes two inputs, and returns a number that weights the contribution of each test point's value. Closer points should have a greater influence on the prediction than further points, so the authors seek a kernel function that returns greater values as the distance between two inputs decreases, and decays rapidly as the distances become larger. While there are many kernel functions available to use in MLKR, the authors use the Gaussian kernel:

$$K(x_i, x_j) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{d(x_i, x_j)}{\sigma^2}} \tag{4.2}$$

MLKR seeks to learn an optimal Mahalanobis distance function (eq 2.3) to replace the distance function in eq 4.2. As in NCA, it optimizes over L, where $M = L^T L$, instead of M to avoid optimizing over a PSD constraint. The authors use the quadratic regression error to penalize predictions that are far from the actual value:

$$\mathbb{L} = \sum_i (y_i - c(x_i))^2 \tag{4.3}$$

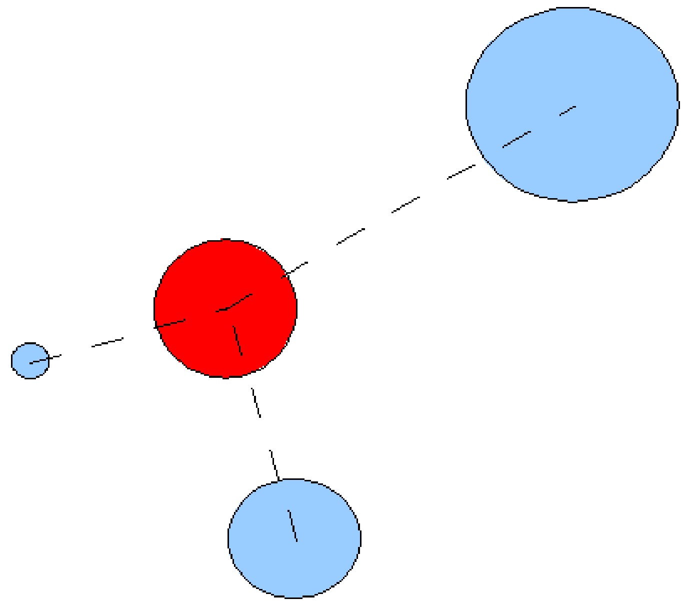


Figure 4.2: The blue circles are training points with value equal to their area. To determine the value of the test red point using kernel regression, we find the weighted sum of each blue point's value, where the weight is determined by a kernel function applied to the distance between the two points.

As in NCA, the error function must exclude the point being optimized over because the kernel function in 4.2 will return large values when comparing x_i to itself, leading to almost perfect predictions in 4.1. This leads to a negligible loss function, which prevents training. Therefore, 4.1 has to be modified to skip comparing points to themselves during training. Optimizing the loss function is performed with a straight-forward gradient method:

$$\Delta A - \epsilon \frac{\partial \mathbb{L}}{\partial L} \tag{4.4}$$

The proof to this solution is shown in [Weinberger and Tesauro, 2007]:

$$\frac{\partial \mathbb{L}}{\partial L} = 4L \sum_i (c(x_i) - y_i) K(x_i, x_j)(x_i - x_j)(x_i - x_j)^T \tag{4.5}$$

While we have not implemented MLKR in our framework, extending our method to incorporate the Mahalanobis matrix from MLKR should be straightforward and virtually identical: first find the closest tangent vectors between all training data points, then learn

MLKR on these vectors instead of on their direct Euclidean difference. To classify a new point, find the tangent vector between the test point and all training points, then apply the learned Mahalanobis metric on this vector when applying 4.1.

Chapter 5

Conclusion

In this thesis, we combine two dissimilar distance functions, both of which theoretically improve on Euclidean distance (for classification), in a way that takes advantage of their specific strengths. A tangent space method derived from tangent distances is first used to bring similar points closer by taking advantage of a priori information of possible transformations that should not affect the classification. This step applies a linear estimate of these transformations on the data points, which arguably more accurately reflects their relationship. The transformed points are then statistically grouped and compared to find a quadratic Mahalanobis metric that is optimized for nearest neighbors classification on a training dataset. We experimented with two different methods to apply this step: NCA and LMNN (we only obtained results from LMNN due to scaling concerns with NCA). Because the transformed points from the tangent step still exist in the original space, any process that optimizes nearest neighbors classification over the vector $x - y$ could be used instead. Because the tangent step transforms data in pairs, this optimization must work on pairs of data points simultaneously - global optimizations (for example, covariance methods) will not work because a point's transformation is dependent on which point it is paired with.

We found that applying a Mahalanobis metric on top of tangent distance leads to only a modest improvement. Tangent distance, a relatively old algorithm, performs quite well on digit classification on its own. Due to potential averaging issues discussed in section 4.3, a single Mahalanobis metric may not be able to differentiate between classes as well as other methods such as Tangent distance or SVM-KNN.

Still, we do see some improvement, which was the original goal. We present a way of combining two very different strategies to offer improvement over the individual methods alone; tangent subspace and Mahalanobis methods are both effective distance functions for classification, and their inherent differences and strengths make them effective in combination. Previous research also backs this assertion. For example, Fraser combines the standard Mahalanobis distance classifier with a second-order estimate of the a priori transformation Manifold in [Fraser et al., 2003b]. Our main contribution is to suggest a new way of combining tangent subspace methods with Mahalanobis methods. Because the performance of tangent distance is still considered impressive, it is useful to find ways of building on top of it. Previous research that combines Mahalanobis metrics with tangent subspace methods have not involved learning the Mahalanobis metrics - instead, they usually involve statistically forming a Mahalanobis metric that fits the data or manifold. This is probably because learning Mahalanobis metrics is a fairly recent innovation. Mahalanobis-metric based learning algorithms have been steadily improving in the past few years [Bar-Hillel et al., 2005] [Xing et al., 2003] [Goldberger et al., 2005] [Weinberger et al., 2005], so we believe this trend may lead to improved results in our method.

References

Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6:937–965, 2005. ISSN 1533-7928.

Andy Fraser, Nick Hengartner, Kevin Vixie, and Brendt Wohlberg. Classification modulo invariance, with application to face recognition. *Journal of Computational and Graphical Statistics*, 12(4):829–852, 2003a.

Andy Fraser, Nick Hengartner, Kevin Vixie, and Brendt Wohlberg. Incorporating invariants in mahalanobis distance based classifiers: Application to face recognition. In *International Joint Conference on Neural Networks*, 2003b.

Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, volume 17, Cambridge, MA, 2005.

Trevor Hastie and Patrice Simard. Learning prototype models for tangent distance. In *Advances in Neural Information Processing Systems*, volume 7, 1995.

Wolfgang Macherey, Daniel Keysers, Jörg Dahmen, and Hermann Ney. Improving automatic speech recognition using tangent distance. In *European Conference on Speech Communication and Technology*, volume 3, 2001.

Ryszard S. Michalski, Robert E. Stepp, and Edwin Diday. A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts. In *Progress in Pattern Recognition*, pages 33–55. North-Holland, 1981.

J. R. Munkres. *Topology: A First Course*. Prentice-Hall, 2nd edition, 2000.

”E. Pekalska, P. Paclk, and R. Duin”. ”a generalized kernel approach to dissimilarity based classification”. *Journal of Machine Learning Research*.

J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, volume 12, 2000.

Patrice Simard, Yann LeCun, and John S. Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems*, volume 5, 1993.

Patrice Simard, Yann LeCun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*. Springer, 1998.

Bharath K. Sriperumbudur and Gert Lanckriet. Metric embedding for nearest neighbor classification. Unpublished.

Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

M. Tipping. The relevance vector machine. In *Advances in Neural Information Processing Systems*, volume 12, 2000.

Kilian Weinberger and Gerald Tesauro. Metric learning for kernel regression. In *International Conference on Artificial Intelligence and Statistics*, pages 608–615. 2007.

Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems*, volume 17, 2005.

D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6, 1997.

E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, volume 15, 2003.

H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.