METRIC LEARNING WITH CONVEX OPTIMIZAITON

Kilian Quirin Weinberger

A DISSERTATION

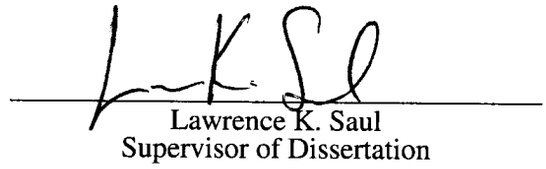
in

Computer and Information Science

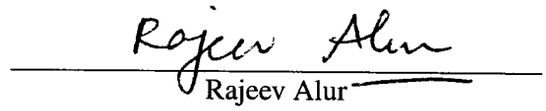
Presented to the Faculties of the University of Pennsylvania in Partial

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2007



Lawrence K. Saul
Supervisor of Dissertation



Rajeev Alur
Graduate Group Chairperson

For

Sabine Schlippe-Weinberger,

who always believed in me,

and for

Lorenz Weinberger,

who taught me scientific thinking.

Acknowledgements

I would like to thank, first and foremost, my advisor Lawrence Saul. Without him, none of the work described in this thesis would have been remotely possible. Lawrence has always supported me with an unlimited source of wisdom, cheerfulness and direction. I can count myself extremely lucky to have had such an amazing supervisor.

I would like to thank my committee Gert Lanckriet, Dan Lee, Ben Taskar and Fernando Pereira for their helpful suggestions. My thesis has benefitted greatly from them.

Further, I would like to thank John Blitzer, Koby Crammer, Fei Sha and Qihui Zhu for their valuable input and contributions to this work. Also, I would like to thank Michael Kearns for his support during the first few years of my PhD and Alina Beygelzimer, Jeff Kephart, Irina Rish and Gerry Tesauro for providing a great environment during my summer internship at IBM. Further, I would like to thank Diane Hu for being helpful at all times.

The last five years in Phialdelphia were some of the best years in my life. This is mostly due to my friends Christie Avraamides, Arvind Bhusnurmath, John Ashley Burgoyne, Marc Corliss, Timothee Cour, Koby Crammer, Nikhil Dinesh, Yuan Ding, Mark Dredze, Aaron Evans, Jihun Hamm, Boulos Harb, Drew Hilton, Liang Huang, Sasha Joseph, Bill Kandylas, Kimia Kashef, Yuanqing Lin, Ameesh Makadia, Ryan McDonald, Andrew McGregor, Angelee Mean, Nick Montfort, Norman Nagl, Antonia Patsialou, Vlad Petric, Georgios E. Fainekos, Pauline Sachar, Ted Sandler, Fei Sha, Marcelo Siqueira, Miranda Stockett, Siddharth Suri, Paul Vernaza, Mirko Visontai, Hanna Wallach, Jenn Wortman, Gary Hui Zhang. I would like to thank especially John Blitzer for his tremendous help at

conferences, before and after the thesis defense and generally all the time.

I also would like to thank my undergraduate tutors Karel Hruda and Martin Powell (who sadly is no longer amongst us) at Oxford University, who shaped my scientific thinking and enabled my postgraduate career. My experience at Oxford was greatly influenced by my friends Edward Botcherby, Oliver Guest, Iwan Lamble, Katharina Lewis, Richard Shelley, Rebecca Wadcock and Adam Williams.

I am very thankful to my family, my parents Lorenz Weinberger and Sabine Schlippe-Weinberger and my two brothers Bastian and Fabian Weinberger, for their strong and lasting support throughout my life. Many thanks also to my grandparents Anna Maria Schlippe and Nikolaus Schlippe and my uncles Oliver Schlippe, Sebastian Schlippe and Richard Weinberger for their constant support throughout my childhood. Without them, I could have never lived the nerdy lifestyle from which I later benefitted so much.

And, most of all, I would like to thank my wife Anne Weinberger Bracy for the many amazing days and months that we have spent together and the many happy years that still lie ahead of us.

Part of the work presented in this thesis is supported by the National Science Foundation under Grant Number 0238323.

ABSTRACT

METRIC LEARNING WITH CONVEX OPTIMIZAITON

Kilian Quirin Weinberger

Lawrence K. Saul

Many machine learning algorithms rely heavily on the existence of a good measure of (dis-)similarity between input vectors. One of the most commonly used measures of dissimilarity is the Euclidean distance in input space. This is often suboptimal in many ways. The Euclidean distance metric does not incorporate any side-information that might be available and it does not take advantage of the data structure or specifics of the machine learning goals. Ideally a metric should be learned for each specific task.

Recent advances in numerical optimization provide us with a powerful tool for metric learning (and machine learning in general): Convex optimization. I will investigate two approaches to metric learning based on convex optimization for two different data scenarios:

The first algorithm, Large Margin Nearest Neighbor (LMNN), operates in a supervised scenario. LMNN learns a metric specifically to improve k-nearest neighbors classification. This is achieved through a linear transformation of the input data that moves similarly labeled inputs close together and separates differently labeled inputs by a large margin. LMNN can be written as a semidefinite program that could be applied to large data sets with up to 60000 training samples.

The second algorithm, Maximum Variance Unfolding (MVU), is designed for an unsupervised scenario. The algorithm finds a low dimensional Euclidean embedding of the data that preserves local distances while globally maximizing the variance. Similar to LMNN, MVU can also be phrased as a semidefinite program. This formulation gives local guarantees and distinguishes the algorithm from prior work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A fundamental challenge of AI is to develop useful internal representations of the external world. The human brain excels at extracting small numbers of relevant features from large amounts of sensory data. Consider, for example, how we perceive a familiar face. A friendly smile or a menacing glare can be discerned in an instant and described by a few well chosen words. On the other hand, the digital representations of these images may consist of hundreds or thousands of pixels. Clearly, there are much more compact representations of images, sounds, and text than their native digital formats. With such representations in mind, this thesis discusses the problem of metric learning—how to detect and represent data efficiently in low dimensions.

For higher-level decision-making in AI, the right representation makes all the difference. We mean this quite literally, in the sense that proper judgments of similiarity and difference depend crucially on our internal representations of the external world. Consider, for example, the images of teapots in Fig. 1.1. Each image shows the same teapot from a different angle. Compared on a pixel-by-pixel basis, the query image and image A are the most similar pair of images; that is, their pixel intensities have the smallest mean-squared-difference. The viewing angle in image B, however, is much closer to the viewing angle in the query image—evidence that distances in pixel space do not support crucial judgments of similarity and difference. A more useful representation of these images would index

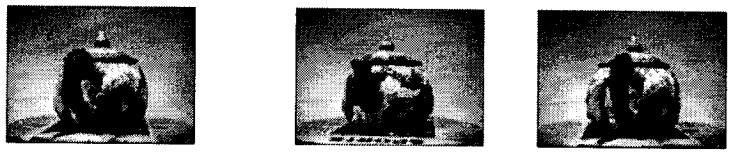


Figure 1.1: Images of teapots: pixel distances versus perceptual distances. As measured by the mean-squared-difference of pixel intensities, image A is closer to the query image than image B, despite the fact that the view in image A involves a full 180 degrees of rotation.

them by the teapot's angle of rotation, thus locating image B closer to the query image than image A.

Objects may be similar or different in many ways. In the teapot example of Fig. 1.1, there is only one degree of freedom: the angle of rotation. More generally, there may be many criteria that are relevant to judgments of similarity and difference, each associated with its own degree of freedom. These degrees of freedom are manifested over time by variabilities in appearance or presentation.

The field of metric learning attempts to distill the modes of variability in a data set and find a representation that reflects these naturally.

Parallel to great progress in machine learning, the past decades have witnessed strong advances in the field of convex optimization. Algorithmic improvements [15], paired with increase in computational power, has brought convex optimization to a level to be able to solve problems of sizes interesting to machine learning researchers and others.

This thesis will explore the possibilities of convex optimization for metric learning. We will introduce two novel algorithms that learn metrics over a given input space. Both algorithms find a new representations of the input data by mapping or embedding the input into a Euclidean vector space. This is carefully done such that the Euclidean distance in this vector space is a "semantically meaningful" measure of dissimilarity (depending on the specifics of the data and task.) Both algorithms are tied together by a joined goal and similar methodology. However, they do cover two different scenarios and operate under

different assumptions.

The first algorithm, *large margin nearest neighbor* (LMNN) operates in a *supervised* scenario: i.e. it assumes that the input data is accompanied with additional label information. LMNN learns a mapping that maps the input vectors into a Euclidean vector space. There, input vectors with similar labels should be close, input vectors with different labels should be separated by a large margin. We will show that under this Euclidean metric a simple kNN classifier is competitive with state-of-the-art multi-class classification algorithms.

The second algorithm, *maximum variance unfolding* (MVU), operates in an *unsupervised* scenario: it does not require that additional label information is available. However, it assumes that the possibly high dimensional input data exhibits fewer degrees of variability than dimensions. The goal is to find an embedding into a Euclidean vector space where distances reflect dissimilarities along these degrees of variability.

We will show that both algorithms, LMNN and MVU can be written as instances of semidefinite programming (SDP). Semidefinite programs are convex optimization problems and therefore have only global minimums and perfectly reproducible results. Also, they can be solved efficiently with polynomial time guarantees [15].

1.1 Some background in machine learning

Machine Learning [59] is a relatively new sub-division of computer science and artificial intelligence. Inspired by humans' capability to learn, the goal of machine learning is to make computers learn from external inputs. One very common approach in this field is *inductive learning* from data. The inductive learning approach uses (large amounts of) sensory input data to extract statistical regularities and patterns in a given data corpus. One can divide the inductive learning methods roughly into three camps: *supervised* and *unsupervised* and *reinforcement learning*. This thesis will concentrate on the first two categories and refer the interested reader to a detailed introduction on *reinforcement learning*

by Kaelbling and Moore [45] or Sutton and Bato [80].

Supervised learning

In *supervised* learning, the available data is assumed to be in pairs $(\vec{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, where $\vec{x}_i$ is regarded as an input vector and y_i as its corresponding label (or also output). The assumption is that the labels y_i are generated from the inputs $\vec{x}_i$ through some function $f(\vec{x}_i) = y_i$. Usually this function f is unknown and it is the task of the machine learning algorithm to infer it from the data. The quality of a supervised algorithm can usually be tested by dividing the available data into two parts. One part is called the *training* data, on which the algorithm is trained, and the second part is the *testing* data, which is only used to measure the quality of the prediction. If the labels are integers from a finite set $\mathcal{Y}$ without any particular ordering, we generally speak of a *classification* problem. Examples of classification algorithms are k-nearest neighbors [21] or support vector machines [23, 70, 86]. Examples of classification problems are handwritten digit recognition [54], document classification [42] or face recognition [84]. If the label is real valued (e.g. $\mathcal{Y} = \mathcal{R}$), we speak of regression problem. Examples of regression algorithms are kernel regression [7] or Gaussian processes [64].

(On a side note, it should be pointed out that this slightly simplified view of machine learning does only cover the most common cases. For example, if the inputs $\vec{x}_i$ represent more complicated inputs with structural dependencies (such as written sentences or sound signals) then more specialized algorithms would be needed [48, 73, 81].)

Unsupervised learning

The main difference between *supervised* and *unsupervised* learning is that the latter does not assume the existence of additional labels. Here, the goal is to model the input data directly, instead of trying to predict the corresponding labels. One of the most common examples of unsupervised learning algorithms is *dimensionality reduction* - the task to find

a low dimensional representation of the data. This can be interesting for several reasons:

- for humans it is very natural to visualize a data set if it is represented in only three (or fewer) dimensions
- it can be easier to interpolate between data points in low dimensional spaces
- further analysis of the data can be much more efficient if it is represented in few dimensions
- data sets often exhibit fewer degrees of variability than the dimensionality of their raw representation
- distances in a carefully selected low dimensional space can be much more meaningful than distances in high dimensional space

In recent years, a lot of work has been done in this area [5, 29, 44, 66, 82, 90, 91, 89, 95]. A substantial part of this thesis will be devoted to dimensionality reduction.

Another area of unsupervised learning is *clustering*. Here, the data is partitioned into different clusters. Examples of clustering algorithms are k-means [10] or spectral clustering [93, 62, 77]. This thesis will not focus on clustering; we refer the interested reader to [2].

1.2 Motivation

During the early stages of Machine Learning, the most commonly used methodologies were largely based on neural networks [10]. Neural networks are very powerful function approximators that lend themselves naturally to supervised learning tasks. The left part of Figure 1.2 shows a schematic layout of a simple (fully-connected) neural net with one input layer and one hidden layer. One appealing aspect of neural nets is that, once their internal structure is specified, they can be treated as a black box learning algorithm. They learn the internal representation of the data (the hidden layers) automatically only based

on feedback from input output pairs. This learning approach is often described as *end-to-end* learning. One big disadvantage of neural networks is that the training involves minimizing a non-convex loss-function [52]. This can cause the optimization to get stuck in local minima and the results are not guaranteed to be reproducible.

During the 1990's, the field of supervised machine learning experienced a change of paradigm with the introduction and great success of kernel based learning algorithms [70, 86]. In comparison to neural networks, kernel based methods are not end-to-end. Here, the outcome is highly dependent on a good choice of a kernel function (or matrix), which encodes pairwise similarities of the input data. The most common kernel based classifiers is the support vector machine (SVM) [70]. One reason for the great success of SVMs was that they demonstrated that a very simple linear classifier can lead to highly accurate results, if it is used in combination with a good measure of dissimilarity (the kernel function). One can interpret the change of paradigm from neural nets to kernel based methods as a shift from sophisticated learning algorithms to sophisticated measures of dissimilarity. The right part of Figure 1.2 illustrates this on a cartoon example. The input vectors (of two different classes) are mapped into a feature space where a simple linear classifier can separated the two classes by a large margin. One great advantage of SVMs is that the training of the classifier is a convex (quadratic) optimization problem. This allows very fast training with reproducible results. On the other hand, one disadvantage of SVMs is that the kernel function has to be chosen a priori.

The first part of this thesis, Chapter 2, is highly inspired by this great success of support vector machines. Similar to SVMs, we will focus on learning a good mapping into a feature space where a simple classifier leads to competitive results with state-of-the-art methods. However, instead of a linear classifier we will use the kNN classification rule [21]. The kNN classifier estimates a label of a test vector by the most common label amongst its k closest training vectors under some predefined metric. Dating back to the 1960s, it is one of the oldest and simplest methods for pattern classification. Nevertheless, it often yields competitive results, and in certain domains, when cleverly combined with

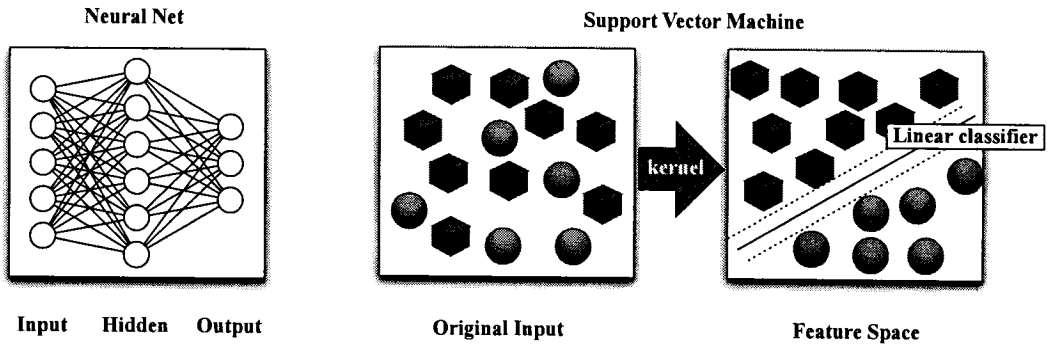


Figure 1.2: A schematic layout of a two layer neural net (left) and the principle behind a support vector machine (right).

prior knowledge has significantly advanced the state-of-the-art [6, 78]. Just as in the training of the SVMs, we will also solve a convex optimization problem. However, instead of a quadratic problem, we will optimize a semidefinite programming problem [85]. Finally (just like SVMs), we will set up the mapping into the feature space such that instances of different classes are separated by a large margin.

The second part of the thesis, Chapter 3, will focus on unsupervised learning. Similar to the change of paradigm in supervised learning, at the unsupervised front (in particular in dimensionality reduction) the field experienced also a shift away from neural networks. The first kernel method for dimensionality reduction was published in 1998 by Schölkopf et al [71]. Until then there existed mainly two ways to reduce the dimensionality of a data set: through linear projections [22, 44] or through non-linear autoassociative neural networks (also referred to as autoencoders) [46]. Autoassociative neural networks (ANN) are trained by using the inputs also as outputs. In other words, the network learns to reproduce its own input. To make the problem interesting, the network structure contains a bottleneck which forces it to learn a low dimensional representation of the input data. The non-linear nature makes ANNs significantly more powerful than any of the linear methods. However, they also inherit the problems typical for neural networks, such as

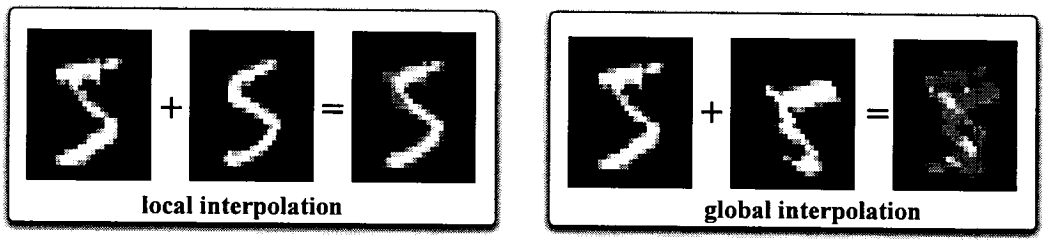


Figure 1.3: An illustration of the difference between local and global interpolation on data sampled from a non-linear manifold (images of handwritten *fives*).

non-reproducability, potential sub-optimal solutions (local minima) and very slow training time. In the year 2000 two publications revolutionized the field of dimensionality reduction. Roweis and Saul [66] and Tenenbaum at al [82] both developed spectral methods for non-linear dimensionality reduction that suffer from none of these problems and scale to large data sets. This sparked great interest in spectral non-linear dimensionality reduction methods and has started the field of *manifold learning* [5, 29, 44, 90, 91, 89, 95].

A *manifold* is a topological space that is locally Euclidean. In manifold learning, the input data is assumed to be sampled from such an underlying *manifold.* For example images of handwritten digits may be represented as high dimensional pixel vectors, however they do lie on a much lower dimensional manifolds determined by degrees of freedom such as thickness of ink, curliness, slant, shift and rotation. Generally, a data set lies on a manifold if locally linear interpolation leads to valid data points inside the set but global interpolation might not. See Figure 1.3 for an illustration. Here we used handwritten *fives* to test local vs global interpolation. Interpolating between two similar images (that are close in Euclidean pixel space) leads to a reasonable image of a *five.* On the other hand, interpolating between two images that are far apart in Euclidean space results only in pixel blur.

We will discuss several dimensionality reduction and manifold learning algorithms in Chapter 3. Further, we will introduce a novel method to learn a kernel matrix for

manifold learning in Chapter 3.1. The proposed method, Maximum Variance Unfolding (MVU) [90, 91], is based on convex optimization. In contrast to Autoassociative networks, it cannot get stuck at sub-optimal solutions and leads to strictly reproducible results. MVU uses the fact that manifolds are locally linear. It "trusts" local Euclidean distances in the raw input space and tries to find a low dimensional Euclidean embedding that preserves these local distances.

Both algorithms, the supervised classifier (LMNN) in Chapter 2 and the unsupervised dimensionality reduction algorithm (MVU) in Chapter 3, learn a mapping of input data into a Euclidean feature space. In the first case, this mapping is geared specifically to improve classification results. In the second case, the goal is to preserve the local structure of the input in a much lower dimensional representation. The goal of both algorithms can be understood to make the Euclidean metric in feature space more useful and "trustworthy". For LMNN, this is explicitly enforced to improve kNN classification. In the case of MVU, the Euclidean distances in feature space are made of locally reliable distances which leads to a globally more meaningful distance measure.

Both LMNN and MVU involve solving a semidefinite program (SDP). SDPs are members of the convex optimization family and can be solved efficiently with global convergence guarantees. Recently, semidefinite programming has sparked great interest in the machine learning community. SDPs lend themselves very well for kernel learning [49, 91] and for the learning of covariance matrices for Mahalanobis metrics [88, 92].

In the remainder of this chapter, we will introduce several mathematical concepts that are necessary to understand following chapters. The math savvy reader is encouraged to skip directly to Chapter 2.

1.3 Mathematical Background

Before we can introduce the various metric learning algorithms, we will quickly review some necessary mathematical background. We will start with the definition and some

properties of positive semi-definite matrices. We will then review the definitions of a metric space and how a well-defined metric over some input can be obtained by mapping it into a Euclidean vector space. Further, we will discuss some background on convex optimization and in particular semidefinite programming. Finally, we will introduce a simple solver that can be used to solve large scale semidefinite programming problems.

1.3.1 Positive Definite Matrices

As already mentioned in previous sections, we will use semidefinite programming as an important tool. Before we are able to talk about SDPs in detail (Section 1.3.3), we will need to define positive semidefinite matrices.

Definition 1 *A symmetric matrix* $\mathrm{M} \in \mathcal{S}^{n\times n}$ *is called* **positive semi-definite** *(we write* $\mathrm{M} \succeq 0$*) if the following property holds:* $\forall \vec{x} \in \mathcal{R}^n, \quad \vec{x}^\top \mathrm{M} \vec{x} \geq 0$.

Definition 2 *If the the condition in Definition 1 holds with strict inequality, then* M *is called* **positive definite** *(we write* $\mathrm{M} \succ 0$*).*

In the remainder of this thesis will denote the set of all positive semidefinite matrices as $\mathcal{S}_+^{n\times n} = \{\mathrm{M} \in \mathcal{S}^{n\times n} \parallel \mathrm{M} \succeq 0\}$. It follows immediately from Definition 1, that a positive semi-definite (PSD) matrix $\mathrm{M} \in \mathcal{S}_+$ has only non-negative eigenvalues. To show this, let $\vec{v}$ be a normalized eigenvector of M with eigenvalue λ. It follows that $\vec{v}^\top \mathrm{M} \vec{v} = \lambda$ and hence λ must be non-negative.

The converse, that any symmetric matrix with only non-negative eigenvalues must be positive semi-definite, can also be shown. Let $\vec{v}_1, \ldots, \vec{v}_n$ be the orthonormal[1] eigenvectors of $\mathrm{M} \in \mathcal{S}$ with eigenvalues $\lambda_1, \ldots, \lambda_n$ where all $\lambda_i \geq 0$. Any vector $\vec{x} \in \mathcal{R}^n$ can be written as a linear combination of the eigenvectors $\vec{x} = \sum_i \alpha_i \vec{v}_i$. It is easy to verify that $\vec{x}^\top \mathrm{M} \vec{x} = \sum_i \alpha_i^2 \lambda_i \geq 0$. Consequently, we can state the following useful lemma:

Lemma 1 *A matrix* $\mathrm{M} \in \mathcal{S}^{n\times n}$ *is positive semi-definite if and only if all its eigenvalues are non-negative.*

[1] Vectors $\vec{v}_1, \ldots, \vec{v}_n$ are orthonormal if and only if for all $i \neq j$ $\vec{v}_i^\top \vec{v}_j = 0$ and $\vec{v}_i^\top \vec{v}_i = 1$.

Recall from linear algebra, that any symmetric matrix can be decomposed into a product of two real matrices $\mathbf{M} = \mathbf{V}\mathbf{\Delta}\mathbf{V}^\top$, where $\mathbf{V}$ contains the orthonormal eigenvectors of $\mathbf{M}$ and the diagonal matrix $\mathbf{\Delta}$ contains the corresponding eigenvalues. As all eigenvalues are non-negative, we can take their square root and arrive at the following lemma:

Lemma 2 *A matrix* $\mathbf{M} \in \mathcal{S}^{n\times n}$ *is positive semi-definite matrix if and only if there exists a matrix* $\mathbf{L} \in \mathcal{R}^{n\times n}$, *such that* $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$.

The proof of Lemma 2 follows from Lemma 1 by setting $\mathbf{L} = \mathbf{V}\sqrt{\mathbf{\Delta}}$ and from definition 2.

1.3.2 Metric Spaces

The goal of this thesis is to introduce new methods to learn metrics that are defined over the input vectors $\vec{x}_1, \ldots, \vec{x}_n \in \mathcal{X}$. This section will introduce some basic definitions and lemmas that form the foundation for the following chapters. The main purpose of this foundation is to allow us to learn a mapping into a Euclidean space to obtain a well defined pseudo-metric over the original inputs.

First of all, we need to specify what we mean by the terms *pseudo-metric* and *metric*:

Definition 3 *A* **pseudo-metric** *over a vector space* $\mathcal{X}$ *is a mapping* $d : \mathcal{X} \times \mathcal{X} \rightarrow R_0^+$ *that satisfies the following properties:* $\forall \vec{x}_i, \vec{x}_j, \vec{x}_k \in \mathcal{X}$

1. $d(\vec{x}_i, \vec{x}_j) + d(\vec{x}_j, \vec{x}_k) \geq d(\vec{x}_i, \vec{x}_k)$ *(triangular inequality)*
2. $d(\vec{x}_i, \vec{x}_j) \geq 0$ *(non-negativity)*
3. $d(\vec{x}_i, \vec{x}_j) = d(\vec{x}_j, \vec{x}_i)$ *(symmetry).*

Definition 4 *A* **metric** *over a vector space* $\mathcal{X}$ *is a* pseudo-metric *satisfying the additional property:* $\forall \vec{x}_i, \vec{x}_j \in \mathcal{X}$

4. $d(\vec{x}_i, \vec{x}_j) = 0 \iff \vec{x}_i = \vec{x}_j$ *(uniqueness).*

In this thesis, we will exclusively use *pseudo-metrics* and (for the sake of better readability) sometimes abuse terminology and refer to them as *metrics*. One of the simplest and most widely used metrics is the Euclidean metric. In machine learning, the Euclidean distance is often used as a measure of dissimilarity (where small distances between instances indicate similarity and large distances indicate dissimilarity). The Euclidean distance between two vectors $\vec{x}_i, \vec{x}_j$ is defined as

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)^\top (\vec{x}_i - \vec{x}_j)}. \tag{1.1}$$

A generalization of the widely used Euclidean distance metric is the family of *Mahalanobis* pseudo-metrics:

Definition 5 *The* **Mahalanobis distance** *over the vector space $\mathcal{R}^n$ with respect to a positive semi-definite matrix $\mathbf{M} \in S_+^{n \times n}$ between two vectors $\vec{x}_i, \vec{x}_j \in \mathcal{R}^n$ is defined as*

$$d_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j)}. \tag{1.2}$$

It is straight-forward to verify that the Mahalanobis distance, parameterized by a semi-positive definite matrix $\mathbf{M}$, is indeed a family of *pseudo-metrics*. The Euclidean metric is a special case with $\mathbf{M} = \mathbf{I}$. Please note that a Mahalanobis distance is only a well-defined *metric* if $\mathbf{M}$ is strictly *positive definite*. For example, one could set $\mathbf{M}$ to the all-zeros matrix (which is trivially positive semi-definite) to show that the condition for a strict *metric* might not hold for all choices of $\mathbf{M}$.

We started out with the intention to learn a metric over the input vectors. The following lemma states that one way to learn a metric is to map the input vectors into a Euclidean vector space and use the Euclidean distance after the mapping.

Lemma 3 *Let $f : \mathcal{X} \to \mathcal{R}^n$ be any well defined mapping and $d : \mathcal{R}^n \times \mathcal{R}^n \to \mathcal{R}_0^+$ be the Euclidean metric over $\mathcal{R}^n$, then $\hat{d} : \mathcal{X} \times \mathcal{X} \to \mathcal{R}_0^+$ defined as $\hat{d}(\vec{x}_i, \vec{x}_j) = d(f(\vec{x}_i), f(\vec{x}_j))$ is a well-defined pseudo-metric over $\mathcal{X}$.*

The proof of Lemma 3 is immediate because Definition 3 holds for all vectors $f(\vec{x}_i), f(\vec{x}_j), f(\vec{x}_k)$, independent of the choice of f. (In fact a much stronger version of Lemma 3 holds if d

is allowed to be any pseudo-metric. On a side note, in order for $\hat{d}$ to be a strict *metric* the function f would need to be required to be *injective*[2].)

With the help of Lemma 3, it is now trivial to show that a Mahalanobis metric is always a well-defined pseudo-metric. Lemma 2 states that the positive semi-definite matrix $\mathbf{M}$ can be decomposed as $\mathbf{M} = \mathbf{L}^\top\mathbf{L}$. If we substitute this decomposition into the definition of the Mahalanobis distance, we can see that the Mahalanobis metric with matrix $\mathbf{M}$ is the Euclidean distance after a linear transformation $f : \vec{x} \rightarrow \mathbf{L}\vec{x}$:

$$d_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = \sqrt{(\mathbf{L}\vec{x}_i - \mathbf{L}\vec{x}_j)^\top(\mathbf{L}\vec{x}_i - \mathbf{L}\vec{x}_j)}. \tag{1.3}$$

Chapter 2 will illustrate how to learn a matrix $\mathbf{M}$ to define a linear function $f : \vec{x} \rightarrow \mathbf{L}\vec{x}$. Chapter 3 will learn the image of a non-linear function f directly without formulating the mapping explicitly.

1.3.3 Semidefinite Programming

In the previous section we showed that it is sufficient to find a mapping into a Euclidean vector space, to obtain a pseudo-metric over the inputs. In some sense, this shifted the focus from learning a metric to learning a mapping. Later on we will show how to formulate the process of learning a mapping as a *semidefinite program* (SDP). This section briefly defines and reviews some basic facts about *semidefinite programs*.

A *semidefinite program* is a convex optimization problem. We try to minimize a convex objective function subject to linear matrix- and a positive semi-definite constraint. In the *standard form*, an SDP over a target matrix $\mathbf{X} \in \mathcal{S}_+^{n\times n}$ and p linear constraints can be written as follows:

minimize$_\mathbf{X}$ **trace**$(\mathbf{CX})$ subject to:	
(1) **trace**$(\mathbf{A}_i\mathbf{X}) = b_i$,	$i = 1, \ldots, p$
(2) $\mathbf{X} \succeq 0$.	

[2]A function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is *injective* if and only if $\forall \vec{x}, \vec{x}' \in \mathcal{X},\ f(\vec{x}) = f(\vec{x}') \Longleftrightarrow \vec{x} = \vec{x}'$.

The matrix $\mathbf{C} \in \mathcal{S}^{n \times n}$ defines the linear objective. The matrices $\mathbf{A}_1, \ldots, \mathbf{A}_p \in \mathcal{S}^{n \times n}$ and scalars $b_1, \ldots, b_p \in \mathcal{R}$ define the p linear equality constraints.

It can easily be shown that by padding the matrix $\mathbf{X}$ with zeros and extending it down the diagonal with variables $\xi_1, \ldots, \xi_p$, we can obtain a *soft-constraint* version of the original problem (assuming that $\mathbf{A}_i, b_i, \mathbf{C}$ are adapted appropriately):

$$\begin{array}{l} \textbf{minimize}_{\mathbf{X}} \ \textbf{trace}(\mathbf{CX}) + \alpha \sum_i \xi_i + \beta \sum_i \mu_i \ \text{subject to:} \\ (1)\ \textbf{trace}(\mathbf{A}_i\mathbf{X}) = b_i + \xi_i - \mu_i, \qquad i = 1, \ldots, p \\ (2)\ \xi_i, \mu_i \geq 0, \qquad i = 1, \ldots, p. \\ (3)\ \mathbf{X} \succeq 0 \end{array}$$

The so called *slack variables* $\xi_1, \ldots, \xi_p$ allow constraints to be slightly violated. The two sums in the objective make sure that the constraints are roughly satisfied by penalizing violations. The two trade-off variables $\alpha, \beta > 0$ determine how much weight the objective should give to the constraint violations. (Please note, that for example $\alpha = 0$ is a simple way to encode inequality constraints.)

1.3.4 Background on Convex Optimization

During the past few decades advances in optimization theory and in cpu processing power provided the mathematical techniques and the hardware to solve convex optimization programs efficiently. SDPs are special instances of convex optimization problems. Several algorithms have been developed that solve SDPs with polynomial time guarantees [15]. For small data sets, these algorithms can be used in form of off-the-shelf packages on standard desktop computers. For large-scale problems, special purpose solvers are generally required.

In machine learning, it is often the case that the real optimization goal (for example classification error on test points) is often a non-convex (or even worse: non-differentiable, non-continuous) function that is too hard to optimize. Therefore, it is common practice, to solve a *surrogate* problem instead, which bounds the original objective. This is an

important point, because this is where the optimization goals of machine learning differs from that of researchers in mathematical optimization. For machine learning applications it is often not crucial to solve an optimization problem up to great accuracy. This is because the objective function is an approximation already. Therefore, a very simple, naïve solver can often lead to surprisingly good results.

In this section, we will briefly review the *gradient alternating projection algorithm* as an example of a widely used method to solve convex optimization problems. This algorithm is particularly popular in the machine learning community because of two reasons: 1. it is very straight-forward to implement, and 2. it scales well to large data sets. We will first introduce the simpler *alternating projection* algorithm and extend it to the *gradient alternating projection* algorithm. For both algorithms, we will provide an intuitive reasoning why it converges to the correct solution.

Alternating projection algorithm

Before we will talk about how to solve the general case of convex optimization problems, let us first concentrate on a much simpler problem. Imagine the matrix C from the previous section is the all-zero matrix. This means that there is no objective function to optimize, and the goal is to find any point within the feasible set (defined by the convex constraints).

To keep the problem formulation as general as possible, let us call the objective variable $\vec{p}$ and the i-th constraint $\vec{p} \in C_i$, where C_i is some convex set (for example a linear hyper-plane). We can then write the optimization problem as follows:

Minimize 0 subject to: $\vec{p} \in \bigcap_{i=1}^{c} C_i$

(where each set C_i is convex)

A common method for solving convex optimization problems of this type is to use the *alternating projection algorithm* [65]. Figure (1.4) illustrates the algorithm on two spheric convex sets. The intuition behind the algorithm is to start at an arbitrary initial point and keep projecting the current point in round-robin fashion onto the convex sets C_i. Let us

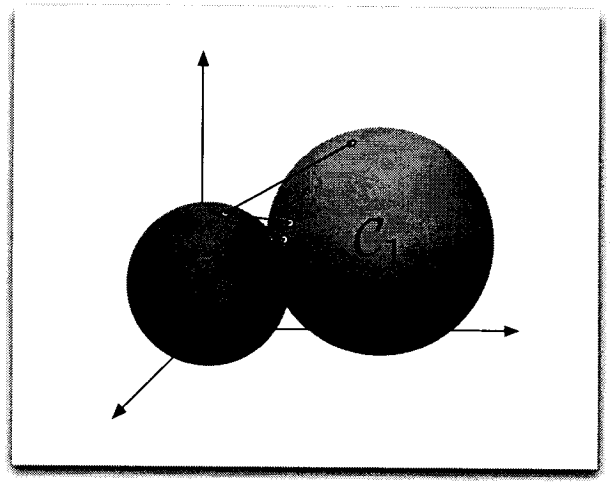


Figure 1.4: The alternating projection algorithm illustrated on two convex sets. Each projection brings the current point closer to the feasible set.

define the projection of a matrix $\vec{p}$ onto a convex set C as $\mathcal{P}_C(\vec{p}) = \text{argmin}_{\vec{p}_* \in C} \|\vec{p} - \vec{p}_*\|_2^2$. **Algorithm 1** shows a pseudo-implementation of the alternating projection algorithm.

This converges because each non-trivial projection (i.e. the vector is not mapped to itself) decreases the L_2 distance to the feasible set (the non-empty intersection of all sets C_i). The proof is immediate from Lemma 6 in this section. Why does the sequence converge to a point within the feasible set? Let $\vec{p}_*$ be the point of convergence. The distance of each projection is zero (as $\vec{p}_*$ gets projected onto itself) and hence $\vec{p}_*$ is an element of every convex set C_i. This implies that $\vec{p}_*$ is an element of the feasible set. For a rigorous proof (including the case of an empty feasible set), please see [16].

Gradient alternating projection

In many cases, we do not only want to find a solution to a set of constraints, but also intend to minimize a convex objective function $f(\vec{p})$. We can state such a problem in the following way:

Algorithm 1 Alternating projection

$\vec{p}_0 = \vec{p}$
$t = 0$
while $t = 0$ or $\|\vec{p}_t - \vec{p}_{t-1}\| > \epsilon$ **do**
  $\vec{p}_{t+1,0} = \vec{p}_t$
  for $i = 1 : c$ **do**
    $\vec{p}_{t+1,i} := \mathcal{P}_{C_i}(\vec{p}_{t+1,(i-1)})$
  end for
  $\vec{p}_{t+1} := \vec{p}_{t+1,c}$
  $t := t + 1$
end while

Minimize $f(\vec{p})$ subject to: $\vec{p} \in \bigcap_{i=1}^{c} C_i$
(where each set C_i is convex, $f(\vec{p})$ is convex and $\frac{\partial f}{\partial \vec{p}}$ exists)

In this case, we can apply the *gradient alternating projection* (GAP) algorithm. The GAP algorithm is identical to the algorithm in the previous section, except that before each round of projections we take a step along the negative gradient of the function $f(\vec{p})$.

Algorithm 2 gives a pseudo code implementation of the gradient alternating projection method. Provided the step-size $\alpha > 0$ is small enough, the algorithm will soon find a feasible solution and then move inside the feasible region to minimize $f(\vec{p})$. If the gradient step moves $\vec{p}_t$ outside of the feasible region, it is projected back.

Algorithm 2 Gradient alternating projection

$\vec{p}_0 = \vec{p}$
$t = 0$
while $t = 0$ or $\|\vec{p}_t - \vec{p}_{t-1}\| > \epsilon$ **do**
  $\vec{p}_{t+1,0} = \vec{p}_t - \alpha \frac{\partial f}{\partial \vec{p}}$
  for $i = 1 : c$ **do**
    $\vec{p}_{t+1,i} := \mathcal{P}_{C_i}(\vec{p}_{t+1,(i-1)})$
  end for
  $\vec{p}_{t+1} := \vec{p}_{+1t,c}$
  $t := t+1$
end while

Semidefinite programs, can only contain two kind of constraints: linear constraints and semi-definite constraints. To use the GAP algorithm to solve SDPs, we will briefly review

how to project a symmetric matrix onto linear half-spaces and onto the convex cone of positive semi-definite matrices.

Projection onto an affine half-space

A linear inequality constraint can be viewed as a constraint of the target vector $\vec{p}$ to lie within an affine half-space $\mathcal{H} = \{\vec{x} \mid \vec{w}^\top \vec{x} \leq c\}$, uniquely defined by some vector $\vec{w}$ and a scalar c. We want to find a closed form solution to assign $\vec{p}_* = \mathcal{P}_\mathcal{H}(\vec{p})$. Of course, if $\vec{p}$ is already in $\mathcal{H}$ then there is nothing left to do and the point is projected onto itself $\vec{p}_* = \vec{p}$. Otherwise, it $\vec{p}$ will always get projected onto the surface of $\mathcal{H}$. This implies that $\vec{p}_*^\top \vec{w} = c$ (we can therefore treat $\mathcal{H}$ also as its surface hyper-plane). The shortest path form $\vec{p}$ onto $\mathcal{H}$ is along the direction $\vec{w}$, which is orthogonal to the surface of $\mathcal{H}$. If we solve the equation $(\vec{p} + \alpha\vec{w})^\top \vec{w} = c$ with respect to α, we obtain the following lemma:

Lemma 4 *Given a vector $\vec{p} \in \mathcal{R}^n$ and a half-space $\mathcal{H} = \{\vec{x} \mid \vec{w}^\top \vec{x} \leq c\}$ or a hyper-plane $\mathcal{H} = \{\vec{x} \mid \vec{w}^\top \vec{x} = c\}$ defined by $\vec{w} \in \mathcal{R}^n$ and $c \in \mathcal{R}$, the projection of $\vec{p}$ onto $\mathcal{H}$ has the closed form solution:*

$$\vec{\mathcal{P}}_\mathcal{H}(\vec{p}) = \begin{cases} p + \frac{c - (\vec{w}^\top \vec{p})}{\|\vec{w}\|_2^2} \vec{w} & \text{if } \vec{p} \notin \mathcal{H} \\ \vec{p} & \text{if } \vec{p} \in \mathcal{H}. \end{cases} \tag{1.4}$$

Projection of a symmetric matrix onto the positive semi-definite cone

The crucial constraints of SDPs is the positive semi-definite constraint. We will now briefly review how to project a matrix $\mathrm{M} \in \mathcal{S}^{n \times n}$ onto the positive semidefinite cone $\mathcal{S}_+^{n \times n}$.

Every symmetric real matrix can be decomposed as $\mathrm{M} = \mathrm{P}\Delta\mathrm{P}^\top$, such that each column of the orthonormal matrix P is an eigenvector of M and the diagonal matrix Δ contains of the corresponding eigenvalues.

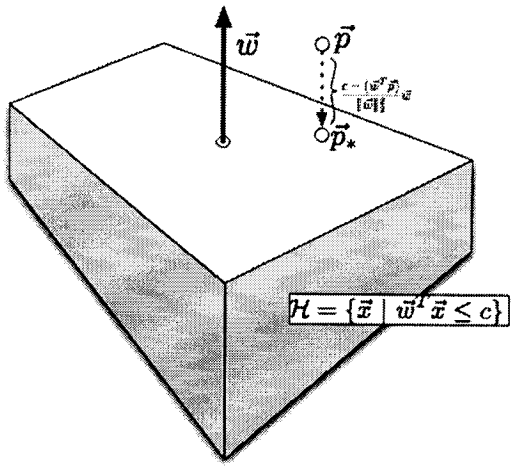


Figure 1.5: A vector $\vec{p}$, with $\vec{w}^\top \vec{p} > c$ is projected onto the half-space $\{\vec{x} \mid \vec{w}^\top \vec{x} \leq c\}$.

Lemma 5 *If* $\mathbf{M} \in \mathcal{S}^{n \times n}$ *is a with eigen-decomposition* $\mathbf{M} = \mathbf{P}\Delta\mathbf{P}^\top$, *the projection onto the cone of positive semi-definite matrices has the closed form solution*

$$\mathcal{P}_{\mathcal{S}_+^{n \times n}}(\mathbf{M}) = \mathbf{P} \max(\Delta, 0)\mathbf{P}^\top, \tag{1.5}$$

where $\max(\Delta, 0)$ *operates element-wise on* Δ.

Sketch of Proof:

Recall that the projection of $\mathbf{M}$ onto $\mathcal{S}_+$ is defined as

$$\mathbf{M}^* = \text{argmin}_{\mathbf{M}^*} \|\mathbf{M} - \mathbf{M}^*\|_F^2. \tag{1.6}$$

Any real symmetric matrix $\mathbf{M}$ can be decomposed as $\mathbf{M} = \mathbf{P}\Delta\mathbf{P}^\top$, where $\mathbf{P}$ is the orthogonal matrix with each column a normalized eigenvector of $\mathbf{M}$ and Δ is a diagonal matrix of the corresponding eigenvalues. Let the positive eigenvalues of $\mathbf{M}$ be $\lambda_1, \ldots, \lambda_p$ and the negative eigenvalues be $-\mu_1, \ldots, -\mu_n$. The matrix $\mathbf{P}$ is orthogonal and describes a rotation. The Frobenius norm is invariant to rotation of the row and column vectors and we can therefore multiply both matrices $\mathbf{M}$ and $\mathbf{M}^*$ with $\mathbf{P}$ from the left and with $\mathbf{P}^\top$ from the right without influencing the result. If we let $\bar{\mathbf{M}}^* = \mathbf{P}^\top \mathbf{M}^* \mathbf{P}$, the resulting objective

becomes

$$\min \|\Delta - \bar{\mathbf{M}}^*\|_F^2 = \min \sum_i (\lambda_i - \bar{\mathbf{M}}^*_{ii})^2 + \sum_j (-\mu_j - \bar{\mathbf{M}}^*_{jj})^2 + \sum_{i \neq j} (\bar{\mathbf{M}}^*_{ij})^2. \quad (1.7)$$

We have the restriction that the matrix $\bar{\mathbf{M}}^*$ is positive semi-definite and one necessary condition is that all diagonal entries are non-negative, i.e. $\bar{\mathbf{M}}^*_{ii} \geq 0$. The objective (1.7) is a sum of squares and we can minimize each of the quadratic terms individually. The first term is clearly minimized if we set every $\bar{\mathbf{M}}^*_{ii} = \lambda_i$. The second term expands to $\mu_j^2 + 2\mu_j \bar{\mathbf{M}}^*_{jj} + (\bar{\mathbf{M}}^*_{jj})^2$, which is a sum of non-negative terms and is therefore minimized when $\bar{\mathbf{M}}^*_{jj} = 0$ (remember that due to the positive-semidefiniteness of $\bar{\mathbf{M}}^*$ we must have that $\bar{\mathbf{M}}^*_{jj} \geq 0$). The last term, involving all the off-diagonal terms, is clearly minimized when $\bar{\mathbf{M}}^*_{ij} = 0$ for all $i \neq j$. The resulting matrix is $\bar{\mathbf{M}}^* = \max(\Delta, 0)$ and the result follows straight from the definition of $\bar{\mathbf{M}}^*$. Q.E.D.

A little discourse on projected distances

The following section contains a lemma that is crucial to prove the convergence of alternating projection algorithms. The lemma states that if any point $\vec{p} \notin \mathcal{C}$ then it is the case that its projection $\mathcal{P}_{\mathcal{C}}(\vec{p})$ is closer to any point in $\mathcal{C}$ under *squared* Euclidean distance. As the feasible set is a subset of every convex set that the algorithm projects on, the current point moves closer to the feasible set with every projection. (Both the lemma and proof are only for reasons of completeness and are not required to understand any remaining parts of this thesis.)

Lemma 6 *Given a convex set $\mathcal{C}$, and two vectors $\vec{p} \notin \mathcal{C}, \vec{q} \in \mathcal{C}$, then the squared Euclidean distance between the projection $\mathcal{P}_{\mathcal{C}}(\vec{p})$ and $\vec{q}$ is less than the distance between the original vector $\vec{p}$ and $\vec{q}$:*

$$\|\vec{p} - \vec{q}\|_2^2 > \|\mathcal{P}_{\mathcal{C}}(\vec{p}) - \vec{q}\|_2^2 \quad (1.8)$$

Sketch of proof:

The proof of Lemma 6 can be easily deduced from Figure 1.6. By applying the rule of

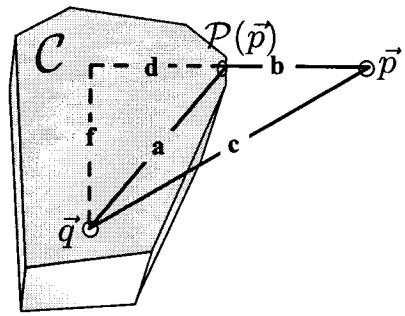


Figure 1.6: A vector $\vec{p}$ is projected onto a convex set $\mathcal{C}$. Clearly $a \leq c$, in fact more precisely $c^2 \geq a^2 + b^2$.

Pythagorus, we obtain $f^2 + (d+b)^2 = c^2$ and $f^2 + d^2 = a^2$. Substituting the second equation into the first results in $c^2 = a^2 + b^2 + 2bd$. Hence it follows that

$$\|\vec{p} - \vec{q}\|_2^2 = \|\vec{p} - \mathcal{P}_{\mathcal{C}}(\vec{p})\|_2^2 + \|\mathcal{P}_{\mathcal{C}}(\vec{p}) - \vec{q}\|_2^2 + 2d\|\vec{p} - \mathcal{P}_{\mathcal{C}}(\vec{p})\|_2^2 \tag{1.9}$$

$$\|\vec{p} - \vec{q}\|_2^2 \geq \|\vec{p} - \mathcal{P}_{\mathcal{C}}(\vec{p})\|_2^2 + \|\mathcal{P}_{\mathcal{C}}(\vec{p}) - \vec{q}\|_2^2 \tag{1.10}$$

$$\|\vec{p} - \vec{q}\|_2^2 > \|\mathcal{P}_{\mathcal{C}}(\vec{p}) - \vec{q}\|_2^2 \tag{1.11}$$

Q.E.D.

We used the assumption that $\mathcal{C}$ is convex indirectly in the definition of d. The top side of the bigger triangle is $b + d$ if $\mathcal{C}$ is indeed convex. Otherwise it could be $b - d$ (if $\angle\vec{q}\mathcal{P}(\vec{p})\vec{p} < \pi$.) The inequality in equation 1.11 is strict because $\vec{p} \notin \mathcal{C}$) and therefore $\|\vec{p} - \mathcal{P}_{\mathcal{C}}(\vec{p})\|_2^2 > 0$.

Chapter 2

Supervised Metric Learning

2.1 Large Margin Nearest Neighbor

2.1.1 Introduction

In this chapter we will introduce a novel metric learning algorithm, *Large Margin Nearest Neighbor* (LMNN), that is especially designed to improve nearest neighbor classification. The k-nearest neighbor (kNN) classification rule is one of the oldest and simplest methods for pattern classification [21]. It classifies each unlabeled example by the majority label amongst its k-nearest neighbors in the training set. Figure 2.1 illustrates this rule on a simple example. Despite its simplicity, it can often yield competitive results and in certain domains, when cleverly combined with prior knowledge, it has significantly advanced the state-of-the-art; see [6, 78]. As kNN determines the test-label exclusively on the basis of the $k-$nearest training points, the quality of the kNN rule depends almost entirely on the quality of the distance metric used to identify those nearest neighbors. If no prior knowledge is available, usually the Euclidean distance measure is used.

Unfortunately, the use of Euclidean distances ignores any statistical regularities that might be estimated from a large training set of labeled examples. Ideally, the metric should be adapted to the particular input vectors and their labels. It can hardly be optimal, for

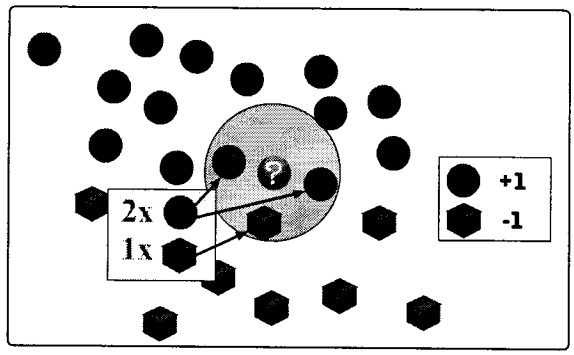


Figure 2.1: An example of 3-nn classification. The green test point (with label '?') will be classified with the most common label amongst its three nearest neighbors +1 (blue).

example, to use the same distance metric for face recognition as for gender identification, even if in both tasks, distances are computed between the same fixed-size images. In fact, a number of researchers have demonstrated that kNN classification can be greatly improved by learning an appropriate distance metric from labeled examples, [19, 35, 74, 76].

Recently, [35] and [74] showed that even a simple linear transformation of the input features can lead to significant improvements in kNN classification. We have shown in Section 1.3.2 that any linear transformation $\vec{x} \rightarrow \mathbf{L}\vec{x}$ gives rise to a well defined pseudo-metric. Given $\mathbf{L}$, we can define the distance between two vectors $\vec{x}$ and $\vec{z}$ to be

$$\mathcal{D}_{\mathbf{L}}(\vec{x}, \vec{z}) = \|\mathbf{L}(\vec{x} - \vec{z})\|^2. \tag{2.1}$$

If we define $\mathbf{M} = \mathbf{L}^\top\mathbf{L}$, we can rewrite the distance metric (2.1) as a Mahalanobis metric under the inverse covariance matrix $\mathbf{M}$:

$$\mathcal{D}_{\mathbf{M}}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})^\top\mathbf{M}(\vec{x} - \vec{z}). \tag{2.2}$$

Mahalanobis metric learning algorithms either learn the matrix $\mathbf{M}$ or the matrix $\mathbf{L}$ to obtain a specialized metric. Which one of the two matrices is learned is entirely a matter of convenience as one defines the other uniquely (up to rotation). It is important to point out that if matrix the $\mathbf{M}$ is learned, it must be constrained to be positive-semidefinite to guarantee that it has a real-valued matrix square-root (As discussed in Section 1.3.2).

In what follows, we describe *large margin nearest neighbor* (LMNN) classificaiton, an algorithm to learn a Mahalanobis metric specifically designed for kNN classification, [88]. Intuitively, the algorithm is based on the obvious insight that if all k nearest neighbors of a test point share the same label as the test point, then the kNN classifier would classify correctly. LMNN learns a linear transformation that attempts to make input neighbors share the same labels. This is achieved by minimizing a loss function that consists of two terms: the first term ensures that neighbors with matching labels are pulled together and the second term pushes neighbors with non-matching labels apart such that they are separated by a large margin. In contrast to our method (which focuses on local neighbors only), previous approaches minimize the pairwise distances between *all* similarly labeled examples and maximize the distance between *all* differently labeled examples, [34, 74, 76, 92]. This objective is far more difficult to achieve and does not leverage the full power of kNN classification, whose accuracy does *not* require that all similarly labeled inputs are tightly clustered.

Our approach is largely inspired by recent work on neighborhood component analysis [35], and metric learning by energy-based models [19]. Despite pursuing similar goals, our method differs significantly as the proposed optimization problem is convex and can be cast as an instance of semidefinite programming. Consequently its global minimum can be efficiently computed.

There are many parallels between our method and classification by support vector machines (SVMs)—most notably, a convex objective function based on the hinge loss, and the potential to work in nonlinear feature spaces by using the "kernel trick". Our framework can be viewed as the logical counterpart to SVMs in which kNN classification replaces linear classification.

Our framework contrasts with classification by SVMs, however, in one intriguing respect: it requires no modification for multiclass problems. Extensions of SVMs to multiclass problems typically involve combining the results of many binary classifiers, or they

require additional machinery that is elegant but non-trivial [23]. In both cases the training time scales at least linearly in the number of classes. By contrast our problem has no explicit dependence on the number of classes.

After we have introduced the LMNN algorithm, we will extend the framework in various ways. One of the most involved extensions is the transition from a single global transformation to several local transformations. We achieve this by partitioning the input space into several parts and learning an individual Mahalanobis metric for each part. However, the kNN classifier demands one function to compare distances consistent throughout the entire input space. Therefore, we combine the local transformations to one global distance function. Informally, the global distance function is a patchwork of many local metrics. To ensure that this function is consistent, we perform all the local training simultaneously in a single objective function. While the training procedure couples the distance metrics in different parts of the input space, the optimization remains a convex problem in semidefinite programming. The globally coupled training of local distance metrics also distinguishes this approach from earlier work, such as [39].

Further, we will also look into combining metric learning with tree data structures [55] and dimensionality reduction. We will also give a brief overview of technical aspects on how to solve the convex optimization problem efficiently.

2.1.2 Notation

In this section we will briefly introduce some necessary terminology that we will use throughout this chapter. We will also use the opportunity to give some intuition behind the algorithm. In the following sections we will then translate this intuition into a convex loss function.

The motivation behind the LMNN algorithm is to learn a linear transformation of the input space to improve the kNN classification of potential test points. However, the test points are not available during training time and we need to "simulate" the test-case. One way of doing this is by minimizing the kNN leave-one-out classification error over the

training set. In other words, we minimize the number of training points that *would have* been missclassified if their label *was* unknown. However, the leave-one-out error is a non-continuous and non-differentiable function and therefore very hard to minimize. Instead, we will construct a loss function which mimics the leave-one-out classification error, but is continuous and well behaved. This auxilliary function, which we will refer to as our *loss function* can then be minimized with standard hill-climbing methods.

Problem setting

Let $\{(\vec{x}_i, y_i)\}_{i=1}^n$ denote a training set of n labeled examples with inputs $\vec{x}_i \in \mathcal{R}^d$ and discrete (but not necessarily binary) class labels y_i. Our goal is to learn a linear transformation $\mathbf{L}: \mathcal{R}^d \rightarrow \mathcal{R}^d$, which gives rise to the following distance metric:

$$\mathcal{D}_{\mathbf{L}}(\vec{x}_i, \vec{x}_j) = \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2. \tag{2.3}$$

Our loss function is based on the observation that for the kNN rule to work reliably, an input $\vec{x}_i$ and its nearest neighbors should share the same label y_i. One can view the loss function as enforcing a protective "shield" around every input. Inside of the shield should only be other inputs with the same label, inputs with different labels are moved outside. We will refer to the "shield" as the *local neighborhood* of an input. The loss function enforces two opposing forces within that region. The first force pulls inputs with matching labels closer. The second force pushes inputs with non-matching labels outside of the local neighborhood.

We begin by developing some useful terminology. In the remainder of this section we will introduce three terms: *target neighbors, local neighborhoods* and *impostors*. For any given input, *target neighbors* are inputs of the same label that we want to be closer, the *local neighborhood* is the local area determining the outcome of the classifier and *impostors* are inputs with different labels that we want to be outside of the *local neighborhoods*. Figure 2.2 illustrates the introduced terms on a particular example.

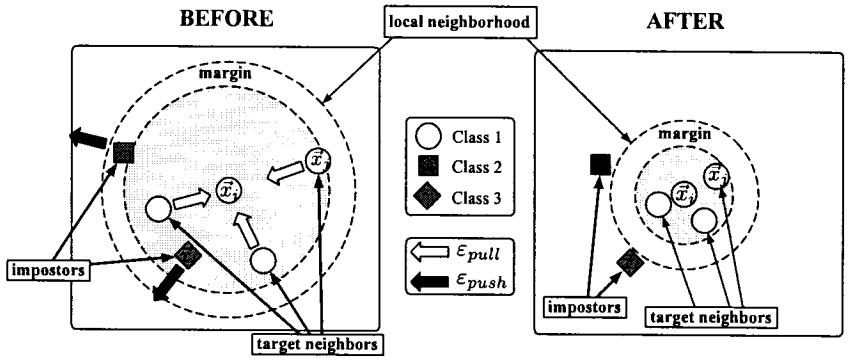


Figure 2.2: A schematic illustration of one input's local neighborhood.

Target neighbors

We intend to have each input vector $\vec{x}_i$ to be surrounded by k nearest neighbors that share the same label y_i. To achieve this goal we identify k vectors for each input $\vec{x}_i$ that share the label y_i. We will refer to these vectors as the *"target neighbors"* of $\vec{x}_i$. The target neighbors are the inputs that we would like to be closest to $\vec{x}_i$ after training.

The target neighbors are fixed a priori to the learning of $\mathbf{L}$. If additional side information is available, they can be hand-picked. In the absence of prior knowledge, they can simply be identified as the k nearest neighbors within the class y_i, determined by Euclidean distance. This was done for all the experiments in this thesis. We use the notation $j \rightsquigarrow i$ to indicate that input $\vec{x}_j$ is a target neighbor of input $\vec{x}_i$. It is important to notice, that the (non-symmetric) relation $j \rightsquigarrow i$ is fixed and does not change during learning.

The first term of the loss function will penalize large distances between any input $\vec{x}_i$ and its target neighbors $\vec{x}_j$. Minimizing the loss function will therefore learn a distance metric under which inputs and their target neighbors are closer together.

Local neighborhood

Each input $\vec{x}_i$ and a corresponding target neighbor $\vec{x}_j$ define a *local neighborhood*. The local neighborhood is like a "shield" or more precisely a hyper-sphere around $\vec{x}_i$ with radius

$\|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 + 1$. If all local neighborhoods of a point $\vec{x}_i$ contain nothing but the target neighbors, the input $\vec{x}_i$ will be classified correctly with the kNN classifier. The constant margin of 1 is important to allow better generalization and robustness. See Figure 2.2 for an illustration of a local neighborhood. Please note that every vector $\vec{x}_i$ and neighbor $\vec{x}_j$ span their own local neighborhood with individual radius.

Impostors

In the same way as the target neighbors of an input $\vec{x}_i$ are encouraged to be pulled closer, there might also be *"impostors"* that we want to push out of the local neighborhoods. Given any input $\vec{x}_i$ with label y_i and a target neighbor $\vec{x}_j$, an impostor is any input $\vec{x}_l$ with a label $\vec{y}_l \neq \vec{y}_i$ such that

$$\|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2 \leq \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 + 1. \tag{2.4}$$

In other words, given the local neighborhood defined by $\vec{x}_i$, its target neighbor $\vec{x}_j$ and a constant margin of size 1, an impostor $\vec{x}_l$ is any input inside that neighborhood with a different label than $\vec{x}_i$. In order for a local neighborhood to contain nothing but target neighbors, we push out all impostors. Figure 2.2 illustrates this concept.

2.1.3 Loss function

With the notation and intuition from the previous section we can now construct a loss function which pulls target neighbors closer and pushes impostors out of the local neighborhoods. These two forces are integrated into the function as two separate terms. We will now discuss each of them independently.

Pulling force

The first term penalizes large distances between each input and its target neighbors. This is achieved by minimizing the squared distances between them after the transformation

$$\varepsilon_{pull}(\mathbf{L}) = \sum_{j \rightsquigarrow i} \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|_2^2. \tag{2.5}$$

It is important to notice that (2.5) only minimizes the distances between similarly labeled *target neighbors* and not between *all* pairs with similar labels. This distinguishes our algorithm from previous work.

Without any counter term, (2.5) could be minimized by setting the matrix $\mathbf{L}$ to the all-zero matrix. It is therefore important that we introduce another force, which prevents all points from collapsing to the origin.

Pushing force

The second term of the cost function penalizes small distances between each input and impostors. Intuitively, target neighbors are pulled closer, while impostors are pushed outside of the local neighborhoods. Let $\vec{x}_i$ be an input vector with target neighbor $\vec{x}_j$. Further, let $\vec{x}_l$ be an impostor (ie $y_l \neq y_i$). See Figure 2.2 for an illustration. The impostor $\vec{x}_l$ is outside of the neighborhood if the following condition holds:

$$\|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|_2^2 \geq 1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|_2^2 \tag{2.6}$$

Intuitively, eq (2.6) makes sense as it ensures that $\vec{x}_l$ is 1 unit further away from $\vec{x}_i$ than $\vec{x}_l$ (in squared distances). Please note that the margin size 1 can be chosen arbitrarily, as all other terms in the loss function will depend on $\mathbf{L}$. If a margin of $c > 0$ was chosen instead, the resulting matrix would be $\frac{1}{c^2}\mathbf{L}$.

The pushing force consists of a cost function that penalizes all instances for which this condition does not hold. In other words, it sums over all violations of the inequality (2.6). To simplify notation, let us introduce a new indicator variable $y_{ij} = 1$ if and only if $y_i = y_j$,

and $y_{ij} = 0$ otherwise. The second part of the loss function, ε_{push} then becomes:

$$\varepsilon_{push}(\mathbf{L}) = \sum_{j \rightsquigarrow i} \sum_{l} (1 - y_{il}) \left[1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2\right]_+ \tag{2.7}$$

where the term $[z]_+ = \max(z, 0)$ denotes the standard hinge loss. The hinge loss enforces the $\geq$ sign from eq (2.6). If the left-hand side is less than the right-hand side the term inside the hinge-loss is negative and the contribution to the loss is 0.

Combined loss function

Now that we have defined the two parts ε_{pull} and ε_{push}, we can combine them in a single loss function. The two terms are weighted with some positive trade-off constant $\mu \in [0, 1]$:

$$\varepsilon(\mathbf{L}) = (1 - \mu)\, \varepsilon_{pull}(\mathbf{L}) + \mu\, \varepsilon_{push}(\mathbf{L}) \tag{2.8}$$

Generally, the parameter μ can be tuned via cross validation. Our experience shows that the loss function (2.8) is not very sensitive with respect to μ and $\mu = 0.5$ seems to work well on most data sets in practice.

Parallels with SVMs

The competing terms in eq. (2.8) are analogous to those in the cost function for SVMs [70].

- In both cost functions, one term penalizes the norm of the "parameter" vector (i.e., the weight vector of the maximum margin hyperplane, or the linear transformation in the distance metric), while the other incurs the hinge loss.
- Just as the hinge loss in SVMs is only triggered by examples near the decision boundary, the hinge loss in eq. (2.8) is only triggered by differently labeled examples that invade each other's neighborhoods.
- Both SVMs and LMNN can be rewritten to access the input vectors only as inner products, which allows the application of the *kernel trick* (see Section 2.1.8).
- In the next paragraph we will show that, similar to SVMs, the loss function (2.8) can also be made convex.

Convex optimization

One downside of the loss function as stated in (2.8) is that it is not convex. Gradient descent methods could get stuck in local minima and results might not be reproducible. Luckily, with a change of variable we can reformulate the optimization of eq. (2.8) as an instance of semidefinite programming (as described in Section 1.3.3 in Chapter 1).

The main step to make our loss function convex is to apply a change of variables from $\mathbf{L}$ to

$$\mathbf{M} = \mathbf{L}^\top \mathbf{L}. \tag{2.9}$$

With the help of (2.9) we can rewrite the distance between two vectors as a *linear* term with respect to $\mathbf{M}$:

$$\mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j). \tag{2.10}$$

Please note that eq. (2.10) is the Mahalanobis distance metric with respect to $\mathbf{M}$ and is equivalent to the Euclidean distance after the mapping $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$. We can now make eq (2.8) convex by substituting in (2.10) for all distances $\mathcal{D}_{\mathbf{L}}$:

$$\varepsilon(\mathbf{M}) = (1-\mu) \sum_{j \rightsquigarrow i} \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) + \mu \sum_{j \rightsquigarrow i} \sum_{l} (1 - y_{il}) \left[1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_l)\right]_+ \tag{2.11}$$

To make sure that (2.11) learns a well defined pseudo-metric, we have to add the constraint that $\mathbf{M}$ is positive semi-definite ($\mathbf{M} \succeq 0$).

So far, we have managed to make the loss function piece-wise linear in $\mathbf{M}$. For it to be a well-defined SDP, we further need to rephrase it as a globally linear function. Please note that the first term (the pulling force) of eq. (2.11) is already linear in $\mathbf{M}$. The second term (the pushing force) contains the hinge-loss, which is only piecewise-linear. Luckily, we can "mimick" the hinge-loss with a soft constraint and penalize its violation. For this we introduce slack variables ξ_{ijl} for any input pairs $\langle i, j \rangle$ with impostor $\vec{x}_l$. The slack variables absorb the violations of the impostor constraints from (2.7). The resulting SDP is then:

Minimize $(1-\mu)\sum_{j\rightsquigarrow i}(\vec{x}_i-\vec{x}_j)^\top \mathbf{M}(\vec{x}_i-\vec{x}_j)+\mu\sum_{j\rightsquigarrow i,l}(1-y_{il})\xi_{ijl}$ **subject to:**

(1) $(\vec{x}_i-\vec{x}_l)^\top \mathbf{M}(\vec{x}_i-\vec{x}_l)-(\vec{x}_i-\vec{x}_j)^\top \mathbf{M}(\vec{x}_i-\vec{x}_j)\ \geq\ 1-\xi_{ijl}$

(2) $\xi_{ijl}\geq 0$

(3) $\mathbf{M}\succeq 0$.

While this SDP can be solved by standard online packages, general-purpose solvers tend to scale poorly in the number of constraints. For this work, we implemented our own special-purpose solver, exploiting the fact that most of the slack variables $\{\xi_{ijl}\}$ never attain positive values. The slack variables $\{\xi_{ijl}\}$ are sparse because most labeled inputs are well separated; thus, their resulting pairwise distances do not incur the hinge loss, and we obtain very few *active* constraints. Our solver was based on a combination of sub-gradient descent in both the matrices $\mathbf{L}$ and $\mathbf{M}$, the latter used mainly to verify that we had reached the global minimum. We projected updates in $\mathbf{M}$ back onto the positive semidefinite cone after each step. Alternating projection algorithms provably converge, [85], and in this case our implementation worked much faster than generic solvers[1]. For a more detailed description of the solver please see Section 2.1.7.

2.1.4 Energy based classificaiton

The most obvious way to perform classification with LMNN is to use the learned Mahalanobis metric for kNN classification. As an alternative, it is also possible to use the objective function (2.8) directly as a energy-based classifier. In this section, we investigate this classifier, inspired by previous work on energy-based models [19]. Energy-based classification of a test example $\vec{x}_t$ is done by finding the label that minimizes the cost function in eq. (2.11). In particular, for a hypothetical label y_t, we compute both parts of (2.11): We accumulate the squared distances to the k nearest neighbors of $\vec{x}_t$ that share the same label in the training set (corresponding to the first term in the cost function); we

[1] A matlab implementation is currently available at http://www.seas.upenn.edu/~kilianw/lmnn.

also accumulate the hinge loss over all pairs of differently labeled examples that result from labeling $\vec{x}_t$ by y_t (corresponding to the second term in the cost function). Finally, the test example is classified by the hypothetical label that minimized the combination of these two terms:

$$y_t = \operatorname{argmin}_{y_t}(1-\mu)\sum_{j \rightsquigarrow t} \mathcal{D}_{\mathbf{M}}(\vec{x}_t, \vec{x}_j) + \mu \sum_{i \rightsquigarrow j, i=t \vee l=t} (1-y_{il}) \left[1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_l)\right]_+ \quad (2.12)$$

Please note that the relation $i \rightsquigarrow j$ depends on the value of y_t. As shown in Fig. 2.5, energy-based classification with this assignment rule generally led to significant reductions in test error rates compared to kNN classification under the Mahalanobis distance metric.

2.1.5 Convergence

In the absence of theoretical results, we tested the LMNN objective (2.8) empirically with synthetically generated data sets from a known uniform distribution. More explicitly, we sampled various training data sets from the rescaled unit-square $X = [0, 10] \times [0, 10]$. We labeled points in the upper half of the square as 1 and those in the lower half as 0. The left plot in Figure 2.3 displays a sample of size $n = 1000$ colored in red and blue, respectively. The domain X is colored according to the decision of a 1-NN classifier. The area that would be classified as 1 is red and the area classified as 0 is blue. The data set is specifically designed such that only the vertical direction is discriminative. Points that lie on the same horizontal level are always of the same class.

The right plot shows the same space after it has been transformed with the LMNN mapping. It can be observed that LMNN mainly rescaled the horizontal dimension by a factor 1.4 and the vertical direction by a factor of 2.2 (in addition to slight skewing). Both directions are stretched to minimize the violations of the margin of size 1. The vertical direction is stretched significantly more than the horizontal direction. This indicates that the loss function learns to put more weight on the meaningful horizontal distances and less weight on vertical distances.

We repeatedly sampled training data from this distribution and each time minimized

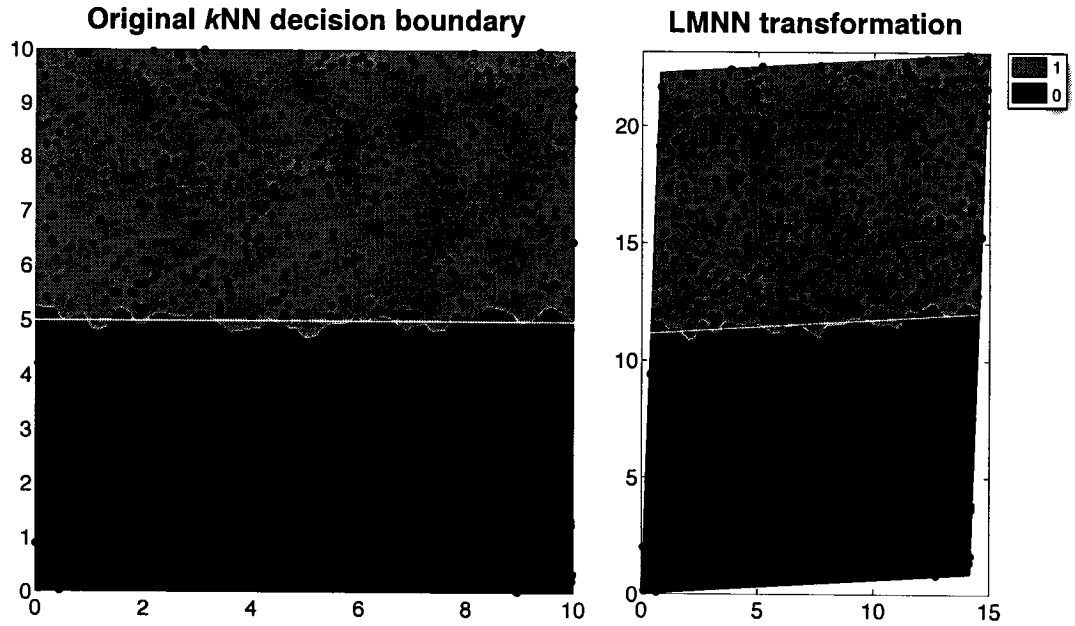


Figure 2.3: A data set of $n = 1000$ data points. See text for details.

eq. (2.8). Given a training set and a transformation L, we evaluated the quality of our learned metric by computing the probability of a uniformly sampled test point to be misclassified with the $1-$NN rule. Let X_u be the upper half or X and X_l be the lower. Further let A_0 be the subset of X in the domain in which the 1-NN rule classifies a test point as 0 (colored blue) and $A_1 = X - A_0$ (colored in red). Let $|X|$ denote the surface area of X. The probability of sampling a test point that will be misclassified is then $p = \frac{|(X_u \cap A_0) \cup (X_l \cap A_1)|}{|X|}$.

We computed the probability p for an increasing amount of uniformly sampled training points. The results, averaged over 100 independent runs per training set size, are plotted in Figure 2.4.

It has been proven that the kNN classifier converges to at most twice the Bayes optimal classification error as the size of the training data grows to infinity [21].[2] In our case, the Bayes optimal classifier is perfect and the probability of misclassification will therefore

[2]This proof requires a well-defined metric and therefore applies to our learned pseudo-metric only if M is of full rank.

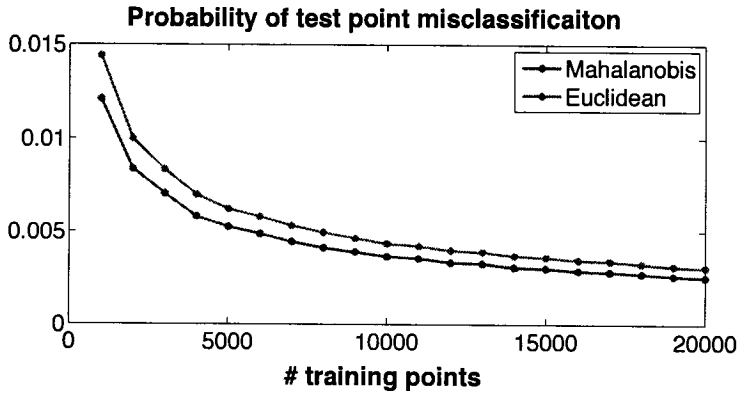


Figure 2.4: The probability of misclassifying a test point with the 1-NN rule with an increasing number of training points.

converge to zero. Figure 2.4 indicates this trend for both metrics as the training size increases. It also shows that the LMNN metric reduces the probability of misclassifying a test point significantly below that of the 1-NN rule under the Euclidean metric.

Finally, we would like to point out that for this synthetic distribution LDA [32] generally recovers the perfect Bayes-optimal decision boundary. However, as we will demonstrate in Section 2.1.6, LMNN clearly outperforms LDA on data sets with more complex (real world) decision boundaries.

2.1.6 Results

We evaluated the large margin nearest neighbors classifier on nine standard data sets of varying size and difficulty. The data domains varied over synthetic data, image, speech, and text data sets. As several of the data sets came in very high dimensional form, we applied PCA as a pre-processing step to reduce computation time and avoid overfitting. Table 2.1 compares the differences of the data sets in detail.

Except for Isolet and MNIST, all of the experimental results are averaged over several runs of randomly generated 70/30 splits of the data. Isolet and MNIST have pre-defined training/test splits. The number of target neighbors (k) was set by cross validation. (For the purpose of cross-validation, the training sets were further partitioned into training and

validation sets.) The weighting parameter (μ) in eqs. (2.11), (2.12) was fixed to $\mu = 0.5$ as the LMNN classifier proved to be surprisingly insensitive to the exact value of μ. We begin by reporting overall trends, then will discuss the individual data sets in more detail.

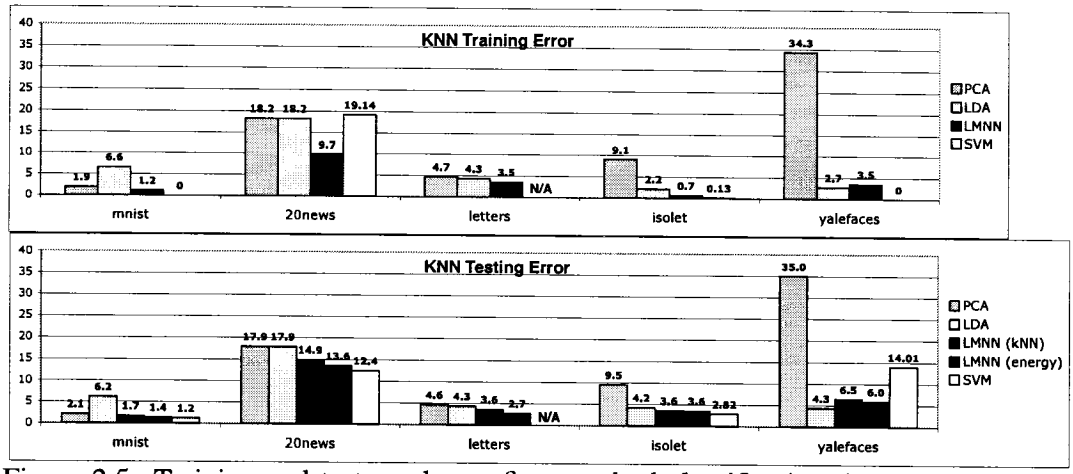


Figure 2.5: Training and test results on five standard classification data sets with kNN classification (under PCA, LDA and the learned Mahalanobis distance) and the Energy classifier.

The results of the experiments on larger data sets are also shown in Fig. 2.5 as well as Table 2.1. All training error rates reported are leave-one-out estimates. We first compare kNN classification error rates using Mahalanobis versus Euclidean distances. To break ties among different classes, we repeatedly reduced the neighborhood size, ultimately classifying (if necessary) by just the $k = 1$ nearest neighbor. Except on the smallest data set (where over-training appears to be an issue), the Mahalanobis distance metrics learned by semidefinite programming led to significant improvements in kNN classification, both in training and testing.

In addition to the kNN classification under Euclidean and Mahalanobis distances, we also applied the energy classifier from Section 2.1.4. From Figure 2.5 it can be observed that the energy classifier outperforms the kNN results under both metrics on all of the large data sets.

In addition to pre-processing with PCA, we also pre-processed the data with LDA [32].

In Figure 2.5 we report the results of kNN classification after the LDA transformation. In Table 2.1, we also show the results of large margin training after LDA. Although pre-processing with LDA helps in some cases (wine, yalefaces), it is generally outperformed by PCA pre-processing and can on occasion lead to drastically worse results (olivetti faces, MNIST).

Finally, we compared our results to those of multiclass SVMs [23]. On each data set (except MNIST), we trained multiclass SVMs using linear, polynomial and RBF kernels and chose the best kernel with cross validation. On MNIST, we used a non-homogeneous polynomial kernel of degree four, which gave us our best results. See also [53]. The results of the energy classifier are very close to the state-of-the-art multi-class SVM results on almost all data sets.

In addition to the results of kNN classification after PCA and LDA transformation and both LMNN classifiers, Table 2.1 also contains the results of various extensions which are discussed in detail in Section 2.1.8. These include re-training LMNN several times consecutively and learning multiple metrics.

In the following paragraphs, we will give a brief but more detailed overview over the individual experiments.

Small data sets with few classes

The wine, iris, and bal data sets are of small size, with less than 500 training examples and just three classes. All three of them originate from the UCI Machine Learning Repository[3]. On data sets of this size, a distance metric can be learned in a matter of seconds. The results in Table. 2.1 were averaged over 100 experiments with different random $70/30$ splits of each data set. Our results on these data sets are roughly comparable (i.e., better in some cases, worse in others) to those of neighborhood component analysis (NCA) and relevant component analysis (RCA), as reported in previous work [35]. These results could potentially be further improved with measures against over-fitting (such as regularization

[3]Available at http://www.ics.uci.edu/~mlearn/MLRepository.html.

or validation). We merely report these results to allow comparisons with previously published work and emphasize that our main focus was on large, more relevant data sets.

Face recognition

The Olivetti face recognition data set[4] contains 400 grayscale images of 40 individuals in 10 different poses. We downsampled the images to 38×31 pixels and used PCA to obtain 200-dimensional eigenfaces [84]. Training and test sets were created by randomly sampling 7 images of each person for training and 3 images for testing. The task involved 40-way classification—essentially, recognizing a face from an unseen pose. Table. 2.1 shows the improvements due to LMNN classification. Fig. 2.7 illustrates the improvements more graphically by showing how the $k = 3$ nearest neighbors change as a result of learning a Mahalanobis metric. (Although the algorithm operated on low dimensional eigenfaces, for clarity the figure shows the rescaled images.)

The (extended) Yale face data set contains $n = 2414$ frontal images of 38 individuals. Of each person there are 64 images which were taken under extreme illumination conditions (a few subjects are represented with fewer images). The top plot of Figure 2.6 shows one example individual with all its 64 images. The input data was sub-sampled and projected onto its leading 200 principal components. We averaged the results over 10 runs of random $70/30$ splits. Due to the large number of classes, we balanced the data set by keeping 45 randomly chosen images of each subject in the training set and 19 in the test set. The training was stopped early when the lowest test error on a hold out validation data set was reached (the validation set consisted of 30% of the training data). Fig. 2.5 shows that the LMNN metric leads to a significantly lower classification error rate than PCA and even outperforms multi-class SVM. The bottom two plots of Figure 2.6 show the eigenfaces [84] of PCA and LMNN. The LMNN eigenfaces are the columns of the L matrix reconstructed in the original input space. One of the biggest source of variability in the data set is the variable illumination, which is independent of the class information.

[4]Available at http://www.uk.research.att.com/facedatabase.html

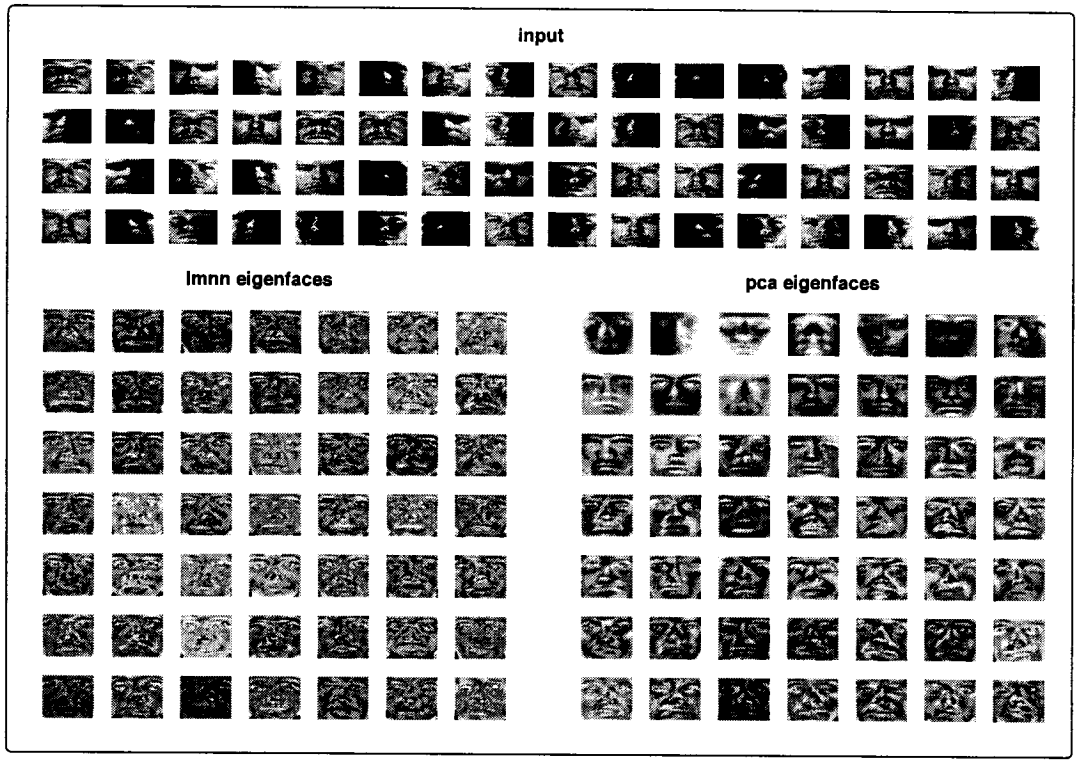


Figure 2.6: Input images and eigenfaces of the yale face dataset. *top:* 64 images of one subject under varying illumination. *bottom:* eigenfaces of LMNN and PCA sorted from top left (leading eigenface) to bottom right.

PCA captures this source of variance in its leading eigenvectors (top two rows of PCA eigenfaces plot). LMNN, on the other hand, focusses only on for classification relevant facial features and has learned to ignore the illumination.

Spoken letter recognition

The Isolet data set from UCI Machine Learning Repository has 6238 examples and 26 classes corresponding to letters of the alphabet. We reduced the input dimensionality (originally at 617) by projecting the data onto its leading 172 principal components—enough to account for 95% of its total variance. On this data set, Dietterich and Bakiri report test error rates of 4.2% using nonlinear backpropagation networks with 26 output units (one per class) and 3.3% using nonlinear backpropagation networks with a 30-bit

Figure 2.7: Images from the Olivetti face recognition data base with nearest neighbors before and after training.

error correcting code [27]. LMNN with energy-based classification obtains a test error rate of 3.7%.

Letter recognition

The letter recognition data set was also taken from the UCI machine learning repository. The data consists of all 26 letters in the English alphabet written in 20 fonts. The resulting images were randomly distorted. The features consist of 16 attributes, such as height, width, correlations of axes and others[5]. Unfortunately we were not able to successfully run the SVM code on the data set (after days of training, it did not terminate on time to include the results). The LMNN energy classifier clearly outperforms the kNN classification rule after PCA, LDA or with the LMNN metric.

[5]Full details on the data set can be found at http://www.ics.uci.edu/~mlearn/databases/letter-recognition/letter-recognition.names

statistics	MNIST	20news	letters	isolet	yalefaces	iris	bal	wine	olivettifaces
dimens.	784	200	16	617	200	4	4	13	200
# inputs	70000	18827	20000	7797	2414	128	535	152	400
# training	60000	13179	14000	6238	1690	90	375	106	280
# testing	10000	5648	6000	1559	724	38	161	46	120
# classes	10	20	26	26	38	3	3	3	40
% validation	0	30	0	0	30	30	0	0	30
# runs	1	10	10	1	10	100	100	100	100
CPU time 1 metric	3h 25m	70m	2m	20.4m	507s	2s	6s	14s	66s
CPU time mul. Metrics	8h 43m	74m	14m	84.2m	829s	5s	8s	16s	149s
# active constr. (1M)	540037	676482	135715	64396	86994	574	41522	10194	3843
# active constr. (MM)	305114	101803	18588	135832	30135	1548	31717	748	70
train (kNN leave-one-out error in %)									
PCA	1.93	18.22	4.67	9.05	34.28	4.20	11.06	24.36	4.20
LDA	6.57	18.22	4.33	2.16	2.73	2.46	10.65	0.26	0.00
PCA-LMNN	1.19	9.73	3.54	0.70	3.54	3.28	9.00	6.12	1.31
LDA-LMNN	6.62	9.72	3.24	1.43	2.32	2.98	9.69	0.23	0.01
MUL-LMNN	1.15	9.62	2.58	0.55	2.73	3.47	7.35	4.77	0.23
MM-LMNN	0.04	7.08	1.55	0.00	3.57	3.13	9.00	6.12	0.68
train (SVM leave-one-out error in %)									
SVM	0.00	19.14	N/A	0.13	0.00	3.36	0.00	0.00	0.00
test (kNN error in %)									
PCA	2.12	17.90	4.63	9.53	34.99	4.87	11.97	25.00	2.80
LDA	6.16	17.90	4.34	4.23	4.34	2.58	11.84	1.98	10.01
PCA-LMNN	1.72	14.91	3.62	3.59	6.48	4.08	9.29	8.39	3.28
LDA-LMNN	6.17	14.90	3.40	4.57	4.91	3.79	10.09	2.11	40.72
MUL-LMNN	1.69	15.08	2.84	3.12	5.83	3.76	7.71	7.30	4.83
MM-LMNN	1.18	14.01	3.20	3.08	6.40	4.37	9.29	8.39	3.11
test (energy error in %)									
PCA	2.39	22.50	3.77	12.31	33.09	3.82	11.71	50.41	2.57
LDA	10.23	22.50	3.36	5.69	4.52	2.63	12.23	2.11	10.01
PCA-LMNN	1.37	13.56	2.67	3.55	5.98	4.00	8.86	7.67	3.16
LDA-LMNN	5.56	13.57	2.33	4.49	5.06	2.82	10.06	2.04	41.93
MUL-LMNN	1.27	14.10	2.38	2.91	5.95	3.34	12.01	7.85	4.87
MM-LMNN	1.23	17.86	2.84	2.99	6.50	3.84	8.88	8.04	3.14
test (SVM error in %)									
SVM	1.20	12.40	1.95	2.82	14.01	3.45	1.92	22.24	1.90

Table 2.1: All results and statistics of all experiments. The table shows the kNN and energy errors after PCA, LDA, LMNN after PCA (PCA-LMNN), LMNN after LDA (LDA-LMNN), multiple applications of LMNN after PCA (MUL-LMNN), LMNN with one meric per class after PCA (MM-LMNN) and support vector machines (SVM). For details see text.

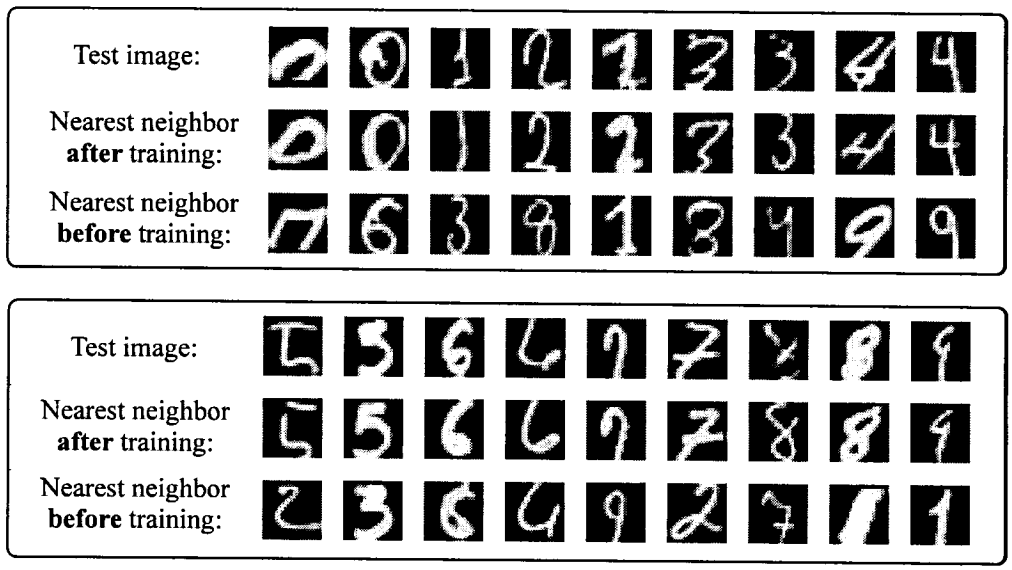


Figure 2.8: Examples of changes within local neighborhoods of handwritten digits before and after learning. (Top: zero-four, Bottom: five-nine.)

Text categorization

The 20-newsgroups data set consists of posted articles from 20 newsgroups, with roughly 1000 articles per newsgroup. We used the 18828-version of the data set[6] which has cross-postings removed and some headers stripped out. We tokenized the newsgroups using the rainbow package [58]. Each article was initially represented by the weighted word-counts of the 20,000 most common words. We then reduced the dimensionality by projecting the data onto its leading 200 principal components. The results in Fig. 2.5 were obtained by averaging over 10 runs with 70/30 splits for training and test data. Our best result for LMMN on this data set at 13.6% test error rate improved significantly on kNN classification using Euclidean distances. LMNN also performed comparably to our best multiclass SVM [23], which obtained a 12.4% test error rate using a linear kernel and 20000 dimensional inputs.

[6] Available at http://people.csail.mit.edu/jrennie/20Newsgroups/

Handwritten digit recognition

The MNIST data set of handwritten digits[7] has been extensively benchmarked [53]. We deskewed the original 28×28 grayscale images, then reduced their dimensionality by retaining only the first 164 principal components (enough to capture 95% of the data's overall variance). Energy-based LMNN classification yielded a test error rate at 1.4%, cutting the baseline kNN error rate by over one-third. Other comparable benchmarks [53] (not exploiting additional prior knowledge) include multilayer neural nets at 1.6% and SVMs at 1.2%. Fig. 2.8 shows some digits whose nearest neighbor changed as a result of learning, from a mismatch using Euclidean distance to a match using Mahalanobis distances. Table 2.1 reveals that the LMNN error can be improved even further by extending the classifier to learn multiple linear metrics. This will be discussed in Section 2.1.8.

2.1.7 Solver

To be able to apply LMNN on large scale problems, we implemented our own special-purpose solver based on the gradient projection rule. By exploiting the specific structure of our particular problem, we could incorporate several crucial speedups that allowed us to apply LMNN to several data sets of sizes up to $n = 60,000$. The solver is an iterative algorithm and in this context, we will refer to the Mahalanobis matrix at iteration t with $\mathbf{M}_t$ and to the distance (2.10) as $\mathcal{D}_t$.

Our solver follows the gradient projection rule (see also Section 1.3.4 in Chapter 1): It first takes a step along the gradient to reduce the loss function and then projects $\mathbf{M}_t$ onto the feasible set. In our case the feasible set is the cone of all positive semidefinite matrices $\mathcal{S}_+$. We will first derive the gradient and will then review the specifics of a projection onto $\mathcal{S}_+$. It is important to point that although we can phrase our optimization as an semidefinite program by modeling the hinge losses with slack variables, in practice it is much faster to leave the hinge-losses in the objective. In this case, the objective is not differentiable at all

[7] Available at http://yann.lecun.com/exdb/mnist/

points. Nevertheless, we will compute the (sub-)gradient and use standard hill-climbing algorithms to find the minimum. It can be shown that this sub-gradient method converges to the right solution, provided the gradient step-size is sufficiently small [15].

Gradient Computation

To simplify the notation, let $\mathbf{C}_{ij} = (\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_j)^\top$. It is straightforward to show that we can express the distances, as defined in equation (2.10), in terms of $\mathbf{M}_t$ and $\mathbf{C}_{ij}$. The distance $\mathcal{D}_t$ then becomes $\mathcal{D}_t(\vec{x}_i, \vec{x}_j) = \mathrm{tr}(\mathbf{M}_t\mathbf{C}_{ij})$. Consequently, we can rewrite the objective (2.11) as follows:

$$\varepsilon(\mathbf{M}_t) = (1-\mu)\sum_{i\rightsquigarrow j}\mathrm{tr}(\mathbf{M}_t\mathbf{C}_{ij}) + \mu\sum_{i\rightsquigarrow j,l}(1-y_{il})\left[1+\mathrm{tr}(\mathbf{M}_t\mathbf{C}_{ij})-\mathrm{tr}(\mathbf{M}_t\mathbf{C}_{il})\right]_+ \quad (2.13)$$

It is important to notice that equation (2.13) is piecewise linear with respect to $\mathbf{M}_t$. Let us define a set of triplets $\mathcal{N}^t$, such that $(i,j,l) \in \mathcal{N}$ if and only if the the indices (i,j,l) trigger the hinge loss in the second part of (2.13). With this definition, we can write the gradient $\mathbf{G}_t$ of $\varepsilon(\mathbf{M}_t)$ as

$$\mathbf{G}_t = \frac{\partial\varepsilon(\mathbf{M}_t)}{\mathbf{M}_t} = (1-\mu)\sum_{\eta_{ij}=1}\mathbf{C}_{ij} + \mu\sum_{(i,j,l)\in\mathcal{N}}(\mathbf{C}_{ij}-\mathbf{C}_{il})\,. \quad (2.14)$$

The set $\mathcal{N}_t$ is potentially very large, which might suggest that the gradient computation is computationally very expensive - computing the outer products $\mathbf{C}_{ij}$ scales quadratic in the input dimension. Luckily, we can exploit the fact that each active triplet (i,j,l) contributes the same amount to the gradient, independent of the value inside the hinge loss. In other words, the only way the gradient changes from one iteration to the next is through the differences between the sets $\mathcal{N}_t$ and $\mathcal{N}_{t+1}$. We can therefore very efficiently express the gradient at iteration $t+1$ with the help of $\mathbf{G}_t$. All we need to do is subtract the contributions of triplets that are no longer active and add the contributions of those that just became active:

$$\mathbf{G}_{t+1} = \mathbf{G}_t - \mu\sum_{(i,j,l)\in\mathcal{N}_t-\mathcal{N}_{t+1}}(\mathbf{C}_{ij}-\mathbf{C}_{il}) + \mu\sum_{(i,j,l)\in\mathcal{N}_{t+1}-\mathcal{N}_t}(\mathbf{C}_{ij}-\mathbf{C}_{il}) \quad (2.15)$$

With a sufficiently small gradient stepsize, the set $\mathcal{N}_t$ changes very little between two consecutive iterations. This makes the computation of (2.15) very fast.

The main computational effort does not go into the computation of the outer products, but into the set $\mathcal{N}_t$. Theoretically, we would have to check the hinge-loss for every triple (i, j, l) (with $i \rightsquigarrow j$), to see if it belongs in $\mathcal{N}_t$ or not. This computation scales $O(kn^2d)$ and would have to be done every iteration. To handle the large computation burden in this case, we adapted an active set method. We exploit the fact that most triplets are never active and maintain a list $\mathcal{N}^{(t)} = \cup_0^t \mathcal{N}$. During most iterations, we only check all triplets in $\mathcal{N}^{(t-1)}$. More precisely, we use the approximation $\mathcal{N}^t \approx \mathcal{N}^t \cap \mathcal{N}^{(t-1)}$. Only every ten or twenty iterations (depending on the fluctuation of $\mathcal{N}_t$) do we check all possible triplets. To ensure convergence to the correct minima, we have to do a final check in the last iteration to see that $\mathcal{N}_t$ did indeed capture all active triplets.

Projection

While minimizing (2.13), we have to ensure that $\mathbf{M}_t$ stays positive semi-definite. To satisfy this constraint, we project the matrix $\mathbf{M}_t$ onto the cone of all positive semidefinite matrices $\mathcal{S}_+$ after each gradient step. This projection can be obtained through a simple diagonalization of $\mathbf{M}_t$. Let $\mathbf{V}\mathbf{\Delta}\mathbf{V}^\top = \mathbf{M}_t$ be the eigendecomposition of $\mathbf{M}_t$, where $\mathbf{\Delta}$ is a diagonal matrix containing the eigenvalues and $\mathbf{V}$ is the orthonormal matrix of corresponding eigenvectors. We can further decompose $\mathbf{\Delta} = \mathbf{\Delta}^- + \mathbf{\Delta}^+$, with $\mathbf{\Delta}^+ = \max(\mathbf{\Delta}, 0)$ contains all positive eigenvalues and $\mathbf{\Delta}^- = \min(\mathbf{\Delta}, 0)$ contains all negative eigenvalues. The projection of $\mathbf{M}_t$ onto the cone of positive semidefinite matrices is

$$\mathcal{P}_{\mathcal{S}}(\mathbf{M}_t) = \mathbf{V}\mathbf{\Delta}^+\mathbf{V}^\top. \tag{2.16}$$

Please see Section 1.3.4 in Chapter 1 for more details and background.

Algorithm

Given the update rules for the gradient (2.15) and the projection (2.16), we can implement our own gradient projection algorithm. (For a more detailed discussion behind the principles of gradient alternating projection please see Section 1.3.4). A simplified pseudo-code implementation is illustrated in Algorithm 3. We denote the gradient step-size by $\alpha > 0$. In practice, we found that a good way to set the step-size is to start with a very small value of α. After each step in which the objective decreased, multiply α by a factor of 1.01. If the objective increased, multiply α by 0.5.

Algorithm 3 A simple gradient projection pseudo-code implementation.

$\mathbf{M}_0 := \mathbf{I}$ {Initialize with the identity matrix}
$t := 0$ {Initialize counter}
$\mathcal{N}^{(0)}, \mathcal{N}_0 := \{\}$ {Initialize active sets}
$\mathbf{G}_0 := (1-\mu) \sum_{\eta_{ij}=1} \mathbf{C}_{ij}$ {Initialize gradient}
while (not converged) **do**
  if $\mathrm{mod}(t, 10) = 0 \vee$ (almost converged) **then**
    compute $\mathcal{N}_{t+1}$ exactly
    $\mathcal{N}^{(t+1)} := \mathcal{N}^{(t)} \cup \mathcal{N}_{t+1}$ {Update active set}
  else
    compute $\mathcal{N}_{t+1} \approx \mathcal{N}_{t+1} \cap \mathcal{N}^{(t)}$ { Only search active set}
    $\mathcal{N}^{(t+1)} := \mathcal{N}^{(t)}$ {Keep active set untouched}
  end if
  $\mathbf{G}_{t+1} := \mathbf{G}_t - \mu \sum_{(i,j,l) \in \mathcal{N}_t - \mathcal{N}_{t+1}} (\mathbf{C}_{ij} - \mathbf{C}_{il}) + \mu \sum_{(i,j,l) \in \mathcal{N}_{t+1} - \mathcal{N}_t} (\mathbf{C}_{ij} - \mathbf{C}_{il})$
  $\mathbf{M}_{t+1} := \mathcal{P}_{\mathcal{S}}(\mathbf{M}_t - \alpha \mathbf{G}_{t+1})$ {Take gradient step and project onto SDP cone}
  $t := t + 1$
end while
Output $\mathbf{M}_t$

2.1.8 Extensions

In this section, we investigate three possible extensions to make the large margin nearest neighbor classifier more powerful. First we will examine the impact of multiple consecutive applications of LMNN on one data set. Further, we will investigate the possibility to learn multiple, locally linear, metrics instead of a single global metric. Finally, we will

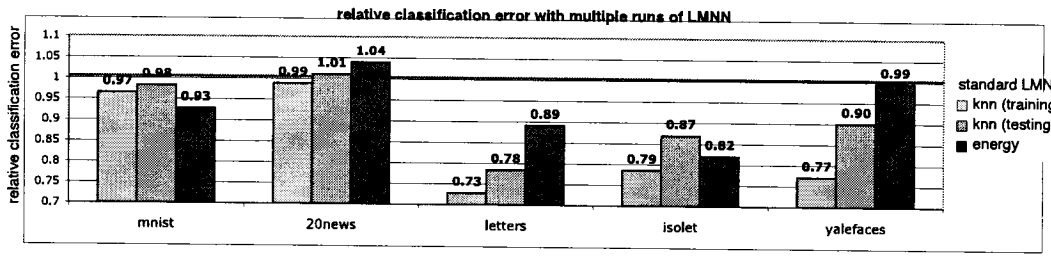


Figure 2.9: The relative kNN and energy classification error after multiple runs of LMNN over a single run of LMNN. The parameter k and the number of runs was chosen with the help of a validation set.

look into the option to kernelize the algorithm and briefly review complementary work by Torresani and Lee [83].

Multiple applications of LMNN

One inherent downside of LMNN can be that the target neighbors have to be picked prior to learning. If outside knowledge is available, this is usually not a big concern. However, in its absence, the best choice is often to use Euclidean distances as an estimate of neighbor relations. One natural question is if we can improve the classification results by re-applying LMNN several times and using the previously learned Mahalanobis distance to pick the target neighbors for the subsequent run. More formally, let $\mathbf{L}_{i-1}$ be the transformation matrix obtained with the $(i-1)^{th}$ run of LMNN. For the i^{th} run, the target neighbors are assigned with the Euclidean distance metric after the linear transformation $\vec{x}_i \rightarrow \mathbf{L}_{i-1}...\mathbf{L}_0\vec{x}_i$ (with $\mathbf{L}_0 = \mathbf{I}$).

Our experience with multiple applications of LMNN is that it can significantly improve the classification results but is prone to overfit. Figure 2.9 shows the kNN and energy classification errors after multiple applications of LMNN relative to that of a single run of LMNN (as described in Section 2.1.6). More explicitly, a value of 1 would indicate that the classification error did not change and a value < 1 indicates an improvement. The absolute results are printed in Table 2.1.

The number of runs and the parameter k were chosen beforehand by applying LMNN

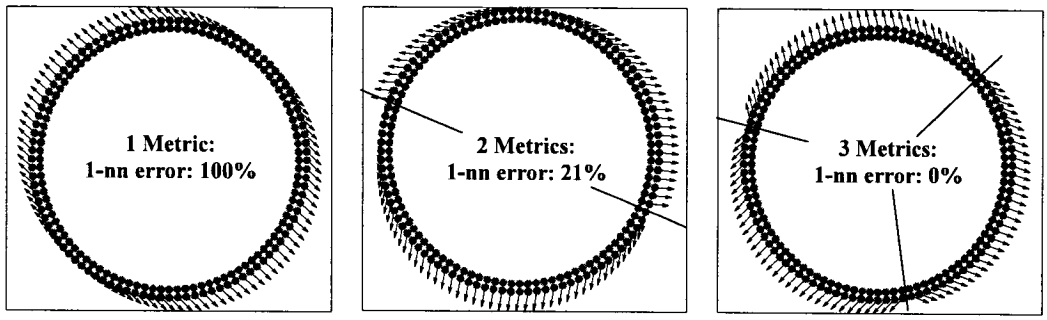


Figure 2.10: A synthetic data set to illustrate the potential of multiple metrics. For details see text.

multiple times on a randomly chosen subset of 70% of the training set. All parameters were set to minimize the classification errors on the remaining 30% of the training set. The data sets are described in detail in Section 2.1.6. It is worth noting that on all of the data sets in Figure 2.9 the results significantly improve or stay almost identical when LMNN is applied multiple times.

Multiple Metrics

On some data sets, it can be the case that the decision boundary of a multi-class data set is too complex and non-linear. In this case, a single linear transformation might not be sufficient to improve the classification results. One interesting extension of LMNN is the introduction of multiple locally linear metrics. Instead of a single global linear transformation, we learn many locally linear transformations.

The idea of learning locally linear distance metrics is already over a decade old, see [39]. However, whereas previous work uses linear discriminant analysis in the local area around a test point, we learn several globally coupled locally linear metrics. Each of these local metrics maximizes their own individual local margins. To make sure that the local metrics are globally comparable, we train all of them simultaneously with a single loss function. Figure 2.10 shows an example of a synthetic data set for which a single

metric is not sufficient. The data set consists of input points sampled from two concentric circles, each of which defines a different class membership. Linear transformations can only stretch or shrink directions within the input space. We measured the leave-one-out training error after learning one, two and three metrics. The images show the principal directions - indicated as arrows - of the individual metrics. A single linear metric cannot capture the non-linear structure of the decision boundary (left). The leave-one-out error is 100%. However, we can cluster the data into several parts and transform each cluster with its own linear mapping. If the data set is divided into two clusters (with k-means), the error rate drops drastically from 100% to 21% (middle). Three clusters are sufficient to separate the two circles enough to obtain a 0% leave-one-out classification error under the 1-nn classification rule (right).

But how should we partition the data into separate clusters? This can be done with a clustering algorithm (such as k-means, or spectral clustering [77]), or according to the label information. Our experience is that in most real world data sets the latter leads to the best classification results. We then minimize a variation of the objective function (2.11) which learns different matrices for each individual cluster. Let us denote the different matrices by $\mathbf{M}^1, \ldots, \mathbf{M}^c$, where c is the number of different local clusters. For simplicity, we will assume that we divide the training data into multiple clusters according to their labels and therefore c coincides with the number of classes. Let y_i be the label of vector $\vec{x}_i$. As the cluster that contains $\vec{x}_i$ is indexed by its label y_i, we can refer to its matrix as $\mathbf{M}^{y_i}$. Further, let us define the cluster-dependent distance between any two vectors $\vec{x}_i$ and $\vec{x}_j$ as:

$$\hat{\mathcal{D}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M}^{y_j} (\vec{x}_i - \vec{x}_j) \tag{2.17}$$

Please note that different from before, this new distance measure $\hat{\mathcal{D}}()$ is no longer symmetric with respect to its inputs. In the following we will "abuse" the term metric slightly by dropping this (for kNN unnecessary) symmetry requirement. With only a few tweaks, we can adapt the original SDP to learn multiple matrices:

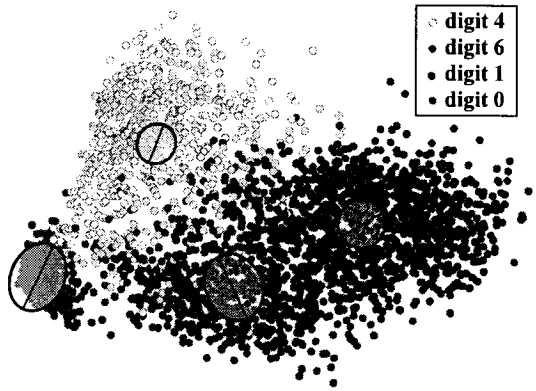


Figure 2.11: Multiple local distance metrics learned for a data set consisting of handwritten digits *four*, *six*, *one* and *zero*.

Minimize $(1-\mu)\sum_{j\rightsquigarrow i}(\vec{x}_i-\vec{x}_j)^\top \mathbf{M}^{y_j}(\vec{x}_i-\vec{x}_j)+\mu\sum_{j\rightsquigarrow i,l}(1-y_{il})\xi_{ijl}$

subject to:

(1) $(\vec{x}_i-\vec{x}_l)^\top \mathbf{M}^{y_l}(\vec{x}_i-\vec{x}_l)-(\vec{x}_i-\vec{x}_j)^\top \mathbf{M}^{y_j}(\vec{x}_i-\vec{x}_j)\ \geq\ 1-\xi_{ijl}$

(2) $\xi_{ijl}\geq 0$

(3) $\mathbf{M}^i\succeq 0$ **for** $i=1,\ldots,c$.

The matrices $\mathbf{M}^i$ are all learned simultaneously in a single semidefinite program. This approach has a clear advantage over the more obvious attempt to learn all metrics independently by applying LMNN independently on each cluster. The distances of a test vector $\vec{x}_t$ to any two training-vectors $\vec{x}_i$ and $\vec{x}_j$ are comparable. See Figure 2.11 for an example of the multiple metrics of a data set consisting of four different types of handwritten digits. The plot shows the first two principal components of the digits *four*, *six*, *one* and *zero*. The metrics are illustrated through superimposed unit circles under the local transformation. The resulting ellipsoids are centered at the class means. The line inside the ellipsoid shows the first principal direction under the corresponding mapping. For visualization purposes, we only trained on the first two principal components of the digit data.

We applied LMNN with one metric per class on all of the data sets from Section 2.1.6. To avoid overfitting, we used a 30% validation data set. Figure 2.12 shows the kNN

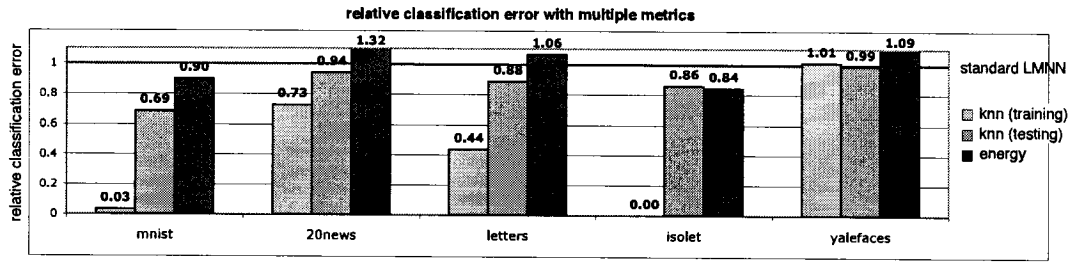


Figure 2.12: The ratio of the $k-$NN and energy classification errors with multiple metrics over a single metric.

(training/testing) and energy error of LMNN with one metric per class relative to the corresponding single-metric error. Table 2.1 shows the absolute results (under "MM-LMNN"). Multiple metrics improve the kNN classification results on several of the large data sets. To speed up training time, we initialized each metric with the single LMNN metric (and added the single metric LMNN training time to the multiple-metrics training time reported in Table 2.1.) On the 20 newsgroups and yalefaces data sets, the multiple metrics started overfitting too quickly and therefore had only little impact on the classification errors. This is not too surprising, as both data sets are of high dimensionality and have many classes which results in a very large number of free parameters. Without validation data set, the training error shrinks to almost zero and the testing error increases. On the MNIST handwritten digits data set, multiple metric LMNN obtained a 1.18% kNN classification error which is slightly lower than multi-class SVMs.

Kernel Version

An alternative way to increase the power of LMNN can be to take advantage of the so-called kernel-trick. This has been investigated in detail by Torressani and Lee [83]. The key idea is to map the input vectors into a higher (possibly infinite) dimensional feature space by applying a feature mapping $\vec{x}_i \rightarrow \Phi(\vec{x}_i)$. To avoid enormous computational costs associated with the high dimensional vectors, the feature vectors are only accessed

through inner products, which are pre-computed in a so-called kernel matrix

$$\mathbf{K}_{ij} = \Phi(\vec{x}_i)^\top \Phi(\vec{x}_j). \tag{2.18}$$

The authors take advantage of the fact that in (2.11) input vectors $\vec{x}_i$ are only accessed in terms of distances (2.10). The key step is to restrict $\mathbf{M}$ to be of the form $\mathbf{M} = \sum_{lm} \mathbf{A}_{lm} \Phi(\vec{x}_l) \Phi(\vec{x}_m)^\top$, where the matrix $\mathbf{A}$ is constraint to be positive semidefinite. Torresani and Lee [83] proved that under these restrictions, the gradient with respect to (2.11) can be written entirely in terms of inner products. This allows the kernel version of LMNN to be executed entirely in kernel space without ever computing a single feature vector. As Torresani and Lee demonstrate, the kernel version of LMNN can lead to significantly better classification results than its original formulation. The kernel version is also significantly slower, as the new target matrix $\mathbf{A}$ scales $O(n^2)$ instead of $O(d^2)$. For details concerning the implementation of the kernel version, please see [83].

LMNN for Dimensionality Reduction

Often it can be useful to have a low dimensional representation of the input data. This can for example be for visualization or to speed up algorithms. In particular, the following section will explore how the nearest neighbor search can be sped up significantly when the training data is mapped into a low dimensional subspace. We will refer a more detailed discussion of the foundations of linear dimensionality reduction to Section 3.1.2.

LMNN can be used for dimensionality reduction in two ways: 1. One can learn a square matrix $\mathbf{L}$ and project the data onto its leading eigenvectors. This is equivalent to applying PCA after the transformation $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$. 2. One can solve the LMNN optimization problem with respect to $\mathbf{L}$ (rather than with respect to $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$) and constrain $\mathbf{L}$ to be rectangular of dimensions $r \times d$, where r is the desired output- and d the input dimensionality. This makes the problem non-convex but, as [83] demonstrated, it leads generally to very good local minima. In the following section, we will compare both methods empirically in the context of tree data structures.

2.1.9 Tree Data Structures

One inherent disadvantage of k-nearest neighbor search is the (relatively) high time complexity at test time. In it's brute-force version, one has to compute the distance between a test point and every single training point to find the k nearest neighbors. This results in a test time-complexity of $O(nd)$ (where n is the number of training points, and d the dimensionality).

One simple way to speed up exact nearest neighbor search is to rotate the input space with PCA such that the dimensions are sorted according to decreasing variance (See Section 3.1.1). During the nearest neighbor query the distance to each training point is computed and compared against the k closest points so far. This distance computation can stop as soon as it exceeds the distance to the (currently believed) $k-th$ closest point. If the most load-bearing dimensions are considered first, the distance computation very often only involves very few dimensions. We will refer to this optimization trick as the naïve method.

Generally there are two major approaches to gain speed-ups for $k-$NN queries. The first is to reduce the dimensionality d. The Johnson-Lindenstrauss Lemma [25] states that n points can be mapped into a space of dimensionality $O(\frac{\log(n)}{\epsilon^2})$ such that the distances between any two points changes only by a factor of $(1 \pm \epsilon)$. This means that often we can reduce the dimensionality of the input data drastically with only very little impact on the nearest neighbor accuracy. Especially for kNN the nearest neighbor computation does not have to be exact, as long as it does not impact the classification error too much.

The second approach involves building a sophisticated tree-based data structure offline before testing. Given a fixed intrinsic dimensionality of the data, this can reduce the nearest neighbor test time complexity in practice to $O(d\log(n))$ [9]. The latter methods work best if the dimensionality of the data is very low.

The ideal approach is to combine both solutions: First reduce the dimensionality of the training data and then build a tree-based nearest neighbor data structure. Figure 2.13 shows the relative speedup gained by dimensionality reduction. We compared the naïve

brute-force search and the use of ball tree data structures in the low dimensional space (we will review ball trees in detail in the following section). The figure illustrates that the extra gain from ball trees reduces drastically with increasing dimensionality.

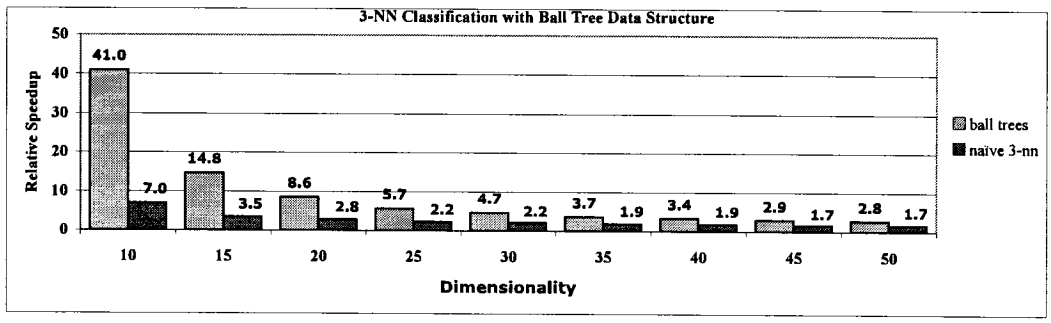


Figure 2.13: The relative speedup on 3-nn classification obtained through dimensionality reduction with and without ball trees (on the MNIST data set with $d = 784$). Ball-trees lead to much higher speed-ups in low dimensions.

In this section we will explore the symbiotic relationship of ball trees with Large Margin Nearest Neighbors for dimensionality reduction. Both methods can benefit from each other: Ball trees can be used to speed up the LMNN training time and LMNN can be used to reduce the dimensionality such that ball trees become more efficient.

Ball Trees Reviewed

Several authors have proposed various tree-based data structures structures to speed up kNN search. Examples are kd-trees [33], ball trees [55, 63] and cover-trees [9]. The main idea behind all these data structures is to partition the training data into a hierarchical bounding shapes. Once set up, the bounding shapes are used in the following way: The distance from the test point to the points inside a bounding shape is guaranteed to be no smaller than the distance from the test point to its surface. If already k points have been found that are closer than this surface, none of the points inside can possibly be the k nearest neighbors. This "pruning" process can enable a significant speedup in nearest neighbor computation time.

We will follow the ball tree data structure as described in [55]. Ball trees are based on the bounding shape principle, with the shapes being hyper-spheres or "balls". Figure 2.14 illustrates this on a cartoon example. If a set S of inputs is encapsulated inside a ball with center $\vec{c}$ and radius r, such that $\forall \vec{x} \in S : \|\vec{x} - \vec{c}\|_2 \leq r$, then for any given test point $\vec{x}_t$ we can bound the distance to any point inside the ball by the following expression:

$$\forall \vec{x}_i \in S \quad \|\vec{x}_t - \vec{x}_i\|_2 \geq \max(\|\vec{x}_t - \vec{c}\|_2 - r, 0). \tag{2.19}$$

A hierarchical data structure is built by splitting the data recursively into two disjoint sets.

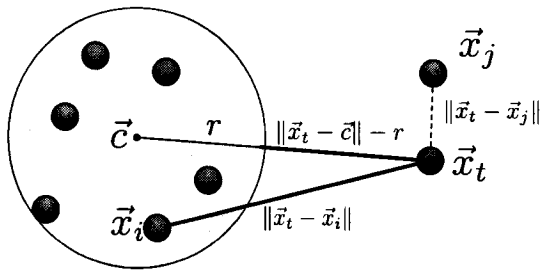


Figure 2.14: The basic principal behind ball trees: For any input $\vec{x}_i$ inside the ball we can bound the distance $\|\vec{x}_t - \vec{x}_i\|_2$ from below using (2.19). If a known input $\vec{x}_j$ is already known to be closer, all inputs inside the ball can be ignored.

Both sets are encapsulated inside hyper-spheres (or "balls") - which might be potentially overlapping. See Figure 2.15 for an illustration. All sets are recursively divided into two until no leaf set contains more than some predefined number of points. Figure 2.15 shows a data set of $n = 1000$ input points that is first divided into two balls (top right). Then each ball is again divided into two more balls (bottom left). This procedure is continued until finally the entire data set is split into 155 balls with each containing no more than 10 inputs.

We split a given data set S into two using the following heuristic: First we select some random input $\vec{r} \in S$ and find the vector $\vec{x}_1 \in S$ that maximizes $\|\vec{x}_1 - \vec{r}\|_2$. We then chose $\vec{x}_2 \in S$ which maximizes $\|\vec{x}_1 - \vec{x}_2\|_2$. We split the data by projecting every point onto the direction $\vec{x}_1 - \vec{x}_2$ and divide at the mean value. (Please note that cutting off at the median would guarantee a tree depth of $\log(n)$ - in practice, however, the mean is significantly

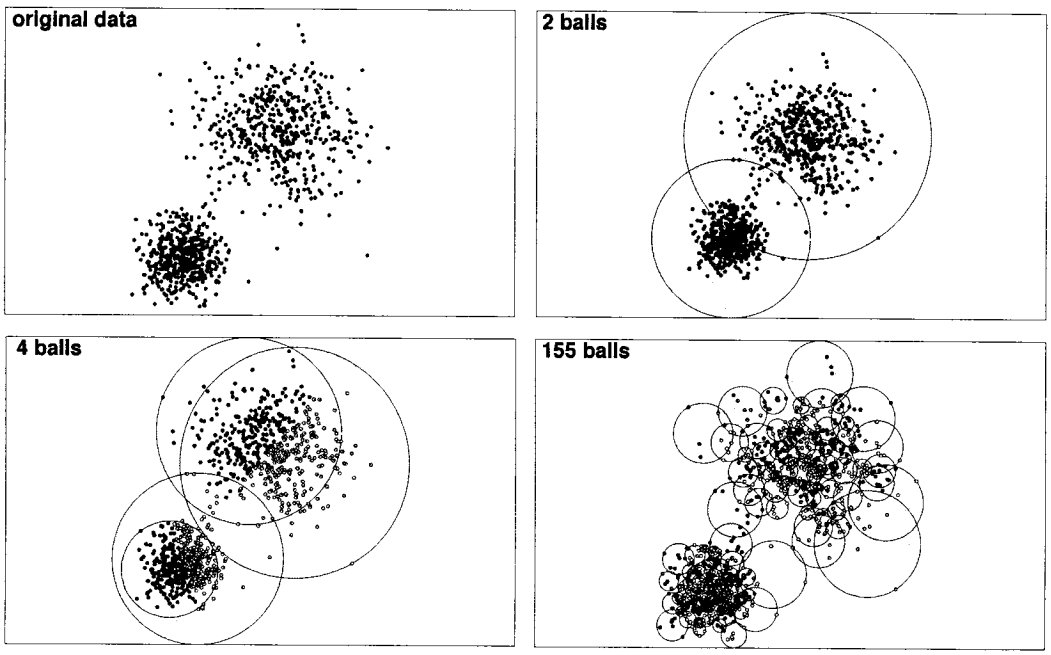


Figure 2.15: A data set sampled from two Gaussian distributions. The figure shows the division into the first 2 and 4 balls and the final 155 balls (where each ball contains at most 10 inputs). Points of the same color belong to the same encapsulating ball. (Each point can only belong to one ball, even if it lies inside the intersection of two or more.)

faster to compute and leads to almost identical results.)

The k nearest neighbors of a test point are found by using a standard depth-first tree-search technique. At each node traverse the child with the closer center point first. Before descending into a sub-tree, use equation (2.19) to check if it can be pruned (if corresponding ball is further than the current best $k-$th nearest neighbor). When a leaf node is reached, compute the distances to all vectors within the set and if necessary update the current best k nearest neighbors.

Note that ball trees recover the exact $k-$nearest neighbors. As pointed out earlier, and as can be observed from Figure 2.13, ball trees will lead to great speed ups mostly with low dimensional data. This is because of the so-called "curse of dimensionality" [41], which makes most balls intersect each other and prevents efficient pruning.

LMNN Training with Ball Trees

The most time consuming part of the LMNN training procedure is the gradient update of the hinge losses from eq. (2.7). It requires a search over all points $j \rightsquigarrow i, l$ to check if the corresponding hinge loss is violated. The solver described in Section 2.1.7 reduces the number of these searches by maintaining an active list of previously violated hinge-losses. Nevertheless, this search is computationally most intensive and scales $O(n^2 d)$.

We can define the problem of finding violated hinge-losses formally as follows: For any given pair of points $\vec{x}_i, \vec{x}_j$ such that $j \rightsquigarrow i$, find all vectors $\vec{x}_l$ with $y_{il} = 0$ such that

$$\|\vec{x}_l - \vec{x}_i\|_2 \leq h_{ij} \tag{2.20}$$

with $h_{ij} = \sqrt{\|\vec{x}_i - \vec{x}_j\|_2^2 + 1}$. Ball trees can speedup this process significantly. The same pruning technique as with the $k-$nn search applies: If for some ball the lower bound from equation (2.7) is already greater than h_{ij}, none of the inputs inside of it need to be considered. As we only need to check for potential points $\vec{x}_l$ that have a different class than $\vec{x}_i$ (ie $y_{il} = 0$), we build one ball tree data structure per class and search for each class separately.

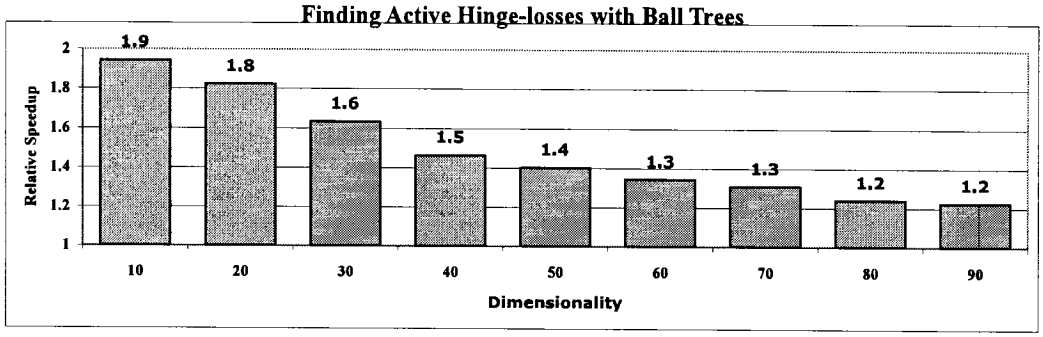


Figure 2.16: The relative speed up from using ball trees to find all active hinge-losses in the MNIST digits data. The benefit of ball trees reduces as the dimensionality of the input increases.

Figure 2.16 shows the relative speed up obtained from ball trees for the active hinge-loss search. Although the impact is not as drastic as for the nearest neighbor search (Figure 2.13), ball trees do speed up the search for active hinge losses significantly. Here we used the MNIST data set after dimensionality reduction with principal component analysis. The reason why the impact of ball trees on the hinge loss search shown in Figure 2.16 is not as high as that of the kNN search in Figure 2.13 is the constant margin of 1. Depending on the scale of the input, this can inflate h_{ij}, which causes a high number of sub-trees to be traversed. This effect can be reduced by multiplying the input data with a large constant before training. This re-scaling of the input only affects the scale of the matrices M, L and has no effect on the classification accuracy.

Ball Trees with LMNN preprocessing

As we have already established, ball trees are a great way to yield very high speedups over the naïve nearest neighbor search. However, only if the input is of reasonably low dimensionality. As the input dimensionality increases, this speed-up diminishes fast. Figure 2.13 illustrates this on the MNIST data set.

The bar plot shows the speed-up of 3-NN search in low dimensional space relative to

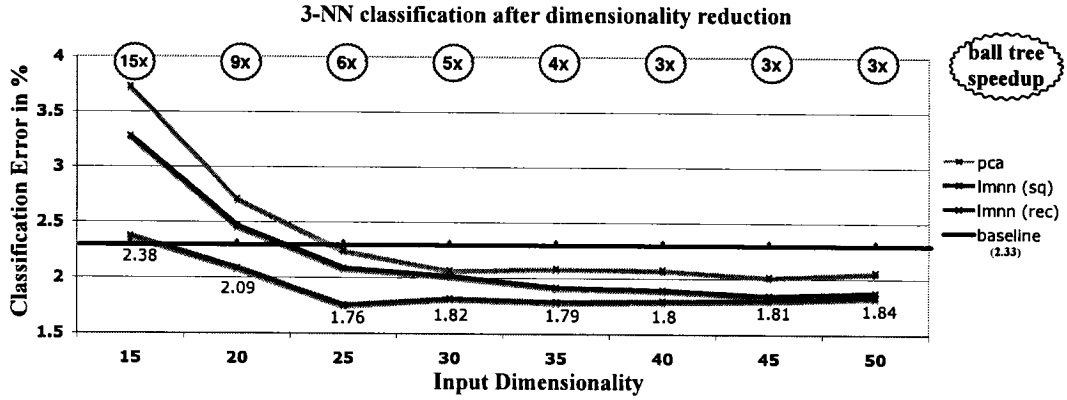


Figure 2.17: The impact of dimensionality reduction on the 3-nn classification error (MNIST data set). We compare PCA and LMNN with a rectangular matrix (rec) and with a square matrix (sq). The speed-up (through ball-trees) is shown on the top of the graph.

the naïve search in the original $d = 784$. The two bars show the speedup of the low dimensional search with ball trees and without. As the dimensionality increases, the extra benefit from ball trees reduces drastically. It is therefore desirable to decrease the dimensionality of the input before using ball trees.

The most commonly used methods for dimensionality reduction for ball trees are random projections, PCA and LDA. None of these methods has an objective geared towards nearest neighbor classification. By projecting the data onto a lower dimensional space, the nearest neighbors change and the classification result can suffer. In the previous section we mentioned two different ways to use LMNN for dimensionality reduction. One way is to learn a transformation and then apply PCA, or to learn a rectangular matrix L in the non-convex setting.

Figure 2.17 compares these two methods with PCA against the nearest neighbor search in the original (high dimensional) input space ("baseline"). The two flavors of LMNN dimensionality reduction are referred to as LMNN (sq) (to learn a square matrix) and LMNN (rec) (to learn a rectangular matrix). The rectangular matrix was of dimensions $r \times 350$, where r varied from 15 to 50. LMNN (sq) was trained with a 350×350 square matrix. The data was projected onto the r leading eigenvectors of the output matrix L.

The experiments in Figure 2.17 were performed on the MNIST handwritten digit data set which was pre-processed with PCA (see Section 2.1.6 for details). The baseline error is the classification error of the raw (high dimensional) input data. The ball-tree speedup is shown on top of the graph. It mostly only depends on the dimensionality, independent of the method of reduction.

The results show that if the input is reduced to 15 dimensions with LMNN (rec) the classification error is only slightly higher (2.38%) than the original raw input baseline (2.33%) - however the classifier is about 14.8 times faster with ball trees. Alternatively one could choose to even reduce the classification error to 1.76% under 25 dimensions and still gain a significant speedup factor of 5.7. Although PCA also leads to lower classification errors than the baseline, both need significantly more dimensions to catch up with the baseline and never outperform LMNN with a rectangular matrix.

In summary, LMNN is a very effective method to pre-process data for fast kNN search with ball trees. Further, it is interesting to see that LMNN (rec) clearly outperforms LMNN (sq), although -as expected- their results slowly converge. Also, it may be surprising to see that LMNN with a rectangular matrix already obtains results very close to those reported in Section 2.1.6 with only 25 dimensions. This shows that only very few dimensions can be necessary to obtain state of the art nearest neighbor classification results.

2.2 Related Work

Before the introduction of LMNN, several algorithms have been proposed to learn linear transformations of the input space. Some of these methods could be used to learn a distance metric for kNN classification. We will first introduce three "traditional" methods based on analysis of second order statistics: PCA, RCA and LDA. All of these three algorithms can be phrased as optimization problems whose global minima can be computed by SVD. We will further review three more recently published algorithms based on convex and non-convex optimization: Mahalanobis Metric for Clustering [92], Pseudo-metric

Online Learning Algorithm [74] and Neighborhood Component Analysis [35].

2.2.1 Traditional Linear Methods

In theory, every linear transformation $\vec{x} \rightarrow \mathbf{A}\vec{x}$ gives rise to a well-defined Mahalanobis metric with matrix $\mathbf{M} = \mathbf{A}^\top\mathbf{A}$. Naturally, this means that there are already many ways to learn Mahalanobis metrics. We will briefly discuss a short list of well known linear feature extraction methods, that can all be viewed as metric learning algorithms: Principal Component Analysis (PCA), Relevant Component Analysis (RCA) and Linear Discriminant Analysis (LDA). All three algorithms use second-order statistics to find signals in the input data to enhance or suppress. PCA projects the input data onto the directions that capture the most amount of variance. RCA uses PCA to de-noise the input and then projects onto directions that enhance the within-class variance. LDA projects onto the directions that best discriminate the different classes of the data. Figure 2.20 shows the results of all three algorithms applied to the same data set. The input was sampled from two Gaussian distributions with identical covariance matrices and shifted means.

Principal component analysis

Given high dimensional data $\vec{x}_1, \ldots, \vec{x}_n \in R^D$, principal component analysis (PCA) (see [44]) finds the projection onto the subspace spanned by the orthogonal directions that contain maximal variability. Let us define the covariance matrix of the input as

$$\mathbf{C} = \frac{1}{m}\sum_{i=1}^{m}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T. \tag{2.21}$$

To find the first direction (1-dimensional projection) with maximum projected variance we will search for a unit vector $\vec{p}$ that is a solution to the following maximization problem

$$\textbf{max } \vec{p}^\top \mathbf{C}\vec{p} \textbf{ subject to: } \vec{p}^\top\vec{p} = 1. \tag{2.22}$$

(The restriction for $\vec{p}$ to be of unit length is necessary to bound the objective without limiting the possible directions of $\vec{p}$.) The optimization (2.22) is not concave but the

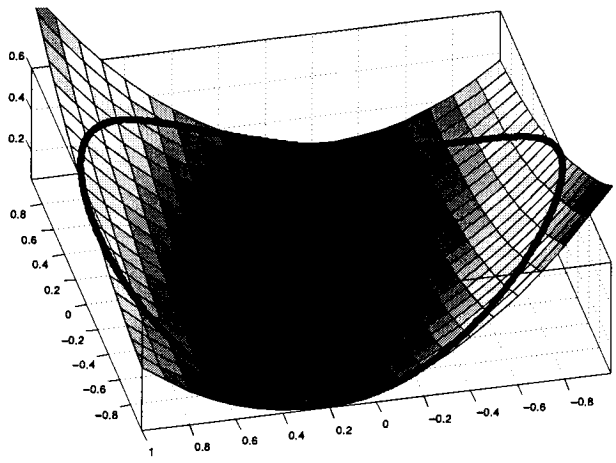


Figure 2.18: An example of the surface of the objective function of PCA as a function of the two dimensional vector $\vec{p}$. The black unit circle illustrates the feasible set satisfying the constraint $\vec{p}^{\top}\vec{p} = 1$. The function is non-convex and has two local minima.

global maxima can be found with SVD. See Fig. 2.18 for a visualization of the objective function. With the help of Lagrange multipliers it is easy to show that (2.22) has a closed form solution consisting of the leading eigenvector of C. The leading eigenvectors of the covariance matrix is generally referred to as the first principal component.

The second principal component is the solution to the same optimization problem as (2.22), however with the remaining co-variance $\mathrm{C} - \vec{p}\vec{p}^{\top}$. It is easy to show that projection onto the d leading eigenvectors of C is the projection onto the $d-$dimensional sub-space with maximum variance. In addition, it can be shown that this projection minimizes the squared reconstruction error.

In Figure 2.20 one can see the result of a synthetic data set of two classes transformed with PCA. It can be seen nicely, that the first PCA principal component reveals the direction with maximum variance.

Strengths: Principal component analysis is an entirely unsupervised method. No label information is necessary. Further, PCA provably finds the low dimensional projection with the minimum L_2 reconstruction error . The computation of the principal components only involves the extraction of the leading eigenvectors of an $n \times n$ matrix, which makes PCA

(relatively) fast in practice.

Limitations: The fact that PCA is entirely supervised can be a weakness in the presence of side information. Even if label information is available it is not incorporated in the feature extraction. For classification purposes, extracting the features that allow the best reconstruction (and capture maximum variance) can be the wrong intuition, see Figure 2.20. Another drawback of PCA is also that the output is not invariant to the scale of the input dimensions.

Relationship with LMNN: PCA is commonly used to improve $k-$NN classification by projecting out low-variance noise. PCA was used for pre-processing on all data sets in this thesis.

Relevant component analysis

A linear projection closely related to PCA is Relevant Component Analysis (RCA) [76]. Different from the unsupervised PCA algorithm, RCA assumes some additional knowledge about which input vectors have similar labels. The data is expected to be accompanied by "chunklet" class membership information. A chunklet is a subset of a class. In other words, if two vectors are in the same chunklet, they share the same class. If two vectors are in different chunklets, it is unknown if they are in the same class or not.

Similar to PCA, relevant component analysis also attempts to project the data into a space of maximum variance, however it only concentrates on directions that are responsible for the variance within the chunklets. These are denoted as the "relevant" features. RCA amplifies the directions responsible for the variance within chunklets.

RCA proceeds in two steps: First PCA is used to filter out the low-variance noise. Then the within-chunklet variance is amplified by modified whitening. It is important that PCA is applied before the whitening step to not also intensify dimensions that carry primarily noise. Intuitively, the PCA step dampens non-informative dimensions and the whitening intensifies informative dimensions that explain the relevant variability of the data.

The whitening transformation of a data set $\vec{x}_1, \ldots \vec{x}_n$ with covariance matrix $\mathbf{C}$ is defined as $\vec{x}_i \rightarrow \mathbf{C}^{-\frac{1}{2}}\vec{x}_i$. It is straight forward to show that the post-whitening covariance matrix is the identity matrix, i.e. $\frac{1}{n}\sum_i \mathbf{C}^{-\frac{1}{2}}\vec{x}_i\vec{x}_i^{\top}\mathbf{C}^{-\frac{1}{2}} = \mathbf{I}$.

Instead of whitening with the global covariance matrix, RCA whitens the data (after applying PCA) with the averaged within-chunklet covariance matrix. Let $C_1, \ldots, C_L$ denote the sets of indices of the L chunklets with means $\vec{\mu}_1, \ldots, \vec{\mu}_L$. The averaged within-covariance matrix is defined as:

$$\mathbf{C}_w = \frac{1}{n}\sum_{l=1}^{L}\sum_{i \in C_l}(\vec{x}_i - \vec{\mu}_l)(\vec{x}_i - \vec{\mu}_l)^{\top}. \tag{2.23}$$

Eq. (2.23) can be understood as a weighted average over all the covariance matrices that are restricted to the individual classes. Let $\mathbf{A}_{PCA}$ be the projection matrix obtained with PCA. We can elegantly combine PCA and the whitening step by mapping the within-covariance matrix onto the leading principal components. Let $\hat{\mathbf{C}}_w = \mathbf{A}_{pca}\mathbf{C}_w\mathbf{A}_{pca}^{\top}$ be the within-class covariance matrix after the projection onto the principal components found by PCA.

We can now express the square-root of the Mahalanobis matrix $\mathbf{A}_{rca}$ as:

$$\mathbf{A}_{rca} = \hat{\mathbf{C}}_w^{-\frac{1}{2}}\mathbf{A}_{pca}. \tag{2.24}$$

Please note that the computation of $\mathbf{A}_{rca}$ can be sped up significantly, if the matrix $\hat{\mathbf{C}}_w$ is decomposed into eigenvectors and eigenvalues and only the diagonal matrix of eigenvalues is raised to the power of $-\frac{1}{2}$. Figure 2.20 illustrates the RCA transformation on a two-class data set. 10% of the data was divided into chunklets of 10 input points of the same class membership. The remaining 90% of the data were unlabeled. It can be observed how RCA whitens the within-class variance to uniform covariance.

Strengths: Relevant component analysis assumes the data to be in "chunklets" of matching class membership. This is a fairly weak assumption. Similar to PCA, the main bottle neck is the eigenvector decomposition of an $n \times n$ matrix.

Limitations: The weak assumption of "chunklet" information also limits the algorithm to be unable to discriminate between different classes. The whitening step amplifies the within-class variability, but if the PCA noise suppression is not chosen carefully, it might also amplify low-variance noise to unit variance.

Relationship with LMNN: Similar to LMNN, RCA also tries to keep points with matching labels close and points with non-matching labels far apart. Also, in contrast to the following algorithms in this chapter, RCA (like LMNN) can be applied to multi-modal data, if separated clusters of a distribution are captured by separate chunklets.

Linear discriminant analysis

If we do have a training data set with precise labels, RCA would not utilize the discriminating information between different classes. With full label information, we can exploit that we know which vectors come from different classes, and separate them explicitly. One of the oldest and still most widely used supervised approaches for feature extraction is Linear Discriminant Analysis (LDA) [32]. LDA finds a linear transformation that minimizes the within-class spread and maximizes the between-class spread. In other words, the intuition behind LDA is that different classes of the data are best identified if the within-class covariance is minimized and the between-class covariance is maximized. See Figure 2.20 for a result obtained with LDA.

Using the same notation as in section 2.2.1, we can define the between class covariance matrix as the covariance matrix induced by the means of the classes

$$\mathbf{C}_b = \frac{1}{L}\sum_{l=1}^{L} \vec{\mu}_l \vec{\mu}_l^{\top} \tag{2.25}$$

and the average within-covariance matrix as $\mathbf{C}_w$ (as defined in section 2.2.1). Linear discriminant analysis finds a projection direction that maximizes the fraction of between-class variance over within-class variance

$$\max_{\vec{p}} \frac{\vec{p}^{\top}\mathbf{C}_b\vec{p}}{\vec{p}^{\top}\mathbf{C}_w\vec{p}}. \tag{2.26}$$

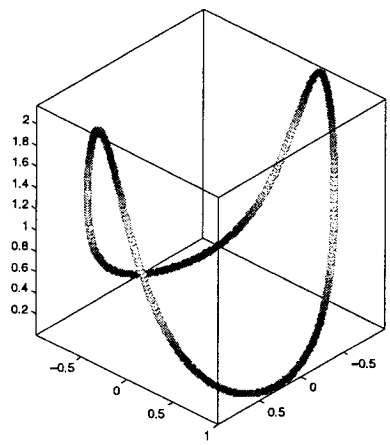


Figure 2.19: An example of the objective function of LDA as a function of the two dimensional vector $\vec{p}$. The plot shows the objective for $\vec{p}^\top \vec{p} = 1$. All extrema are eigenvectors of (2.27).

It is important to notice that (2.26) is invariant to the scale of $\vec{p}$ or the dimensions of the input (we will fix it to $\vec{p}^\top \vec{p} = 1$). Similar to PCA, this objective is not concave but the global maxima can be found with SVD. For an illustration see Figure 2.19. It is maximized when $\vec{p}$ is a solution to the generalized eigenvector problem: $\mathbf{C}_b \vec{p} = \lambda \mathbf{C}_w \vec{p}$.

Similarly to PCA, we can generalize this for more than one dimension by projecting onto the leading n eigenvectors of

$$\mathbf{M} = \mathbf{C}_w^{-1} \mathbf{C}_b. \tag{2.27}$$

Please note that (2.27) is only well defined if $\mathbf{C}_w$ is non-singular. This can be a potential problem when the data is very high dimensional and sparsely sampled. To obtain a projection onto a lower dimensional vector space, we can decompose $\mathbf{M} = \mathbf{A}^\top \Delta \mathbf{A}$ and erase all but the first d columns of $\mathbf{A}$. Figure 2.20 compares LDA with PCA on a data set sampled from two different Gaussian distributions. The principal component of PCA maximizes the projected variance, the projection direction of LDA best separates the two classes. Figure 2.20 illustrates this on a simple synthetic data set.

Strengths: In the case of two normally distributed classes with identical covariance,

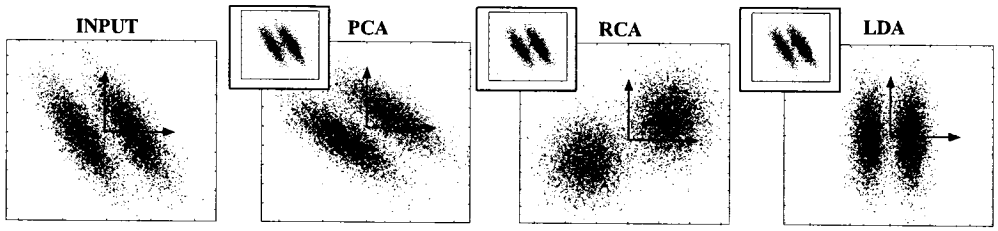


Figure 2.20: N=1000 input points sampled from two Gaussian distributions with identical covariance but different means under PCA, RCA and LDA transformations. The little images on the top left of the projections show the input data with the projection directions super-imposed.

the first LDA principal component can be shown to recover the direction orthogonal to the optimal bayes classifier. Also, similar to PCA and RCA, LDA can be computed very efficiently.

Limitations: If the input is sampled from muldimodal distributions, minimizing the within class spread can be the wrong intuition. LDA does not focus on the local decision boundaries but on the overall variance. As mentioned before, if the data is high dimensional and sparsely sampled, the matrix $\mathbf{C}_w$ can be singular.

Relationship with LMNN: LDA and LMNN are based on similar intuitions. Both algorithms try to bring similarly labeled points close together, and dissimilarly labeled points apart. However, as LMNN focusses on kNN classification, it applies the hinge-loss to only separate dissimilarly labeled points by a large margin. Further, similarly labeled points are moved closer only if they are originally close. LDA has no local ambitions and attempts to put *all* similarly labeled points close together and *all* dissimilarly labeled points apart. This makes LDA much faster, but is not optimized for a local $k-$NN classifier. LDA can be used as an effective pre-processing tool for LMNN (see section 2.1.6).

2.2.2 Recent work on Mahalanobis metric learning

In addition to methods based on second-order statistics, we also discuss three recent algorithms that learn dissimilarity metrics with the help of optimization problems: Mahalanobis Metric for Clustering (MMC) [92], Pseudo-metric Online Learning Algorithm (POLA) [74] and Neighborhood Component Analysis (NCA) [35]. MMC minimizes a convex optimization problem to directly learn the matrix for the Mahalanobis metric that maximizes distances between differently labeled inputs and keeps distances between similarly labeled inputs close. POLA follows this intuition, but processes the data in an online fashion and introduces a margin to separate similarly from differently labeled input pairs. Both POLA and MMC learn well-suited metrics for clustering applications. NCA is an algorithm directly aimed for k-NN classification tasks that uses a stochastic neighborhood assignment to obtain a continuous objective function.

Mahalanobis metric for clustering

Following a similar intuition as linear discriminant analysis, [92] introduced a way to learn a Mahalanobis Metric for Clustering (MMC). Just like LDA, the objective is to minimizes the distances between similarly labeled inputs and maximizes the distances between differently labeled inputs. Just like LMNN, the matrix $\mathbf{M} = \mathbf{A}^\top \mathbf{A}$ is learned with a semi-definite program.

Following previous notation, we denote the input to be $\vec{x}_1, \ldots, \vec{x}_n$ with labels $y_1, \ldots, y_n$. Further, let us define $y_{ij} = 1$ if $y_i = y_j$ and $y_{ij} = 0$ otherwise. Also let $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$. If we write the constraint that the target matrix $\mathbf{M}$ is semi-positive definite as $\mathbf{M} \succeq 0$, we can state the optimization problem of MMC as :

Maximize $\sum_{i,j,y_{ij}=0} \sqrt{\vec{x}_{ij}^\top \mathbf{M} \vec{x}_{ij}}$ **subject to:**

(1) $\sum_{i,j,y_{ij}=1} \vec{x}_{ij}^\top \mathbf{M} \vec{x}_{ij} \leq 1$

(2) $\mathbf{M} \succeq 0$.

The optimization maximizes distances between pairs of inputs with different labels ($y_{ij} = 0$). To make the problem feasible, it constraints the sum over all squared distances of input pairs with similar labels ($y_{ij} = 1$) below 1. Please note that the square root in the objective is essential for MMC to lead to different results than LDA. One side effect is that the optimization is *not* scale-invariant. As the scale of M is learned, the upper bound in the first constraint can be chosen arbitrarily to be 1. Any other bound would change the resulting matrix M only by a positive factor.

One optional simplification of the algorithm is to force the Mahalanobis matrix M to be diagonal. In this case, M is semi-positive definite if and only if all diagonal entries are non-negative. This simplification reduces the number of parameters drastically and makes the optimization problem a linear program.

One extension is to use the kernel trick [70] to perform MMC in nonlinear feature spaces. Similar to the kernelization of LMNN (see Section 2.1.8), the key step is to express the Mahalanobis matrix M in terms of outer products ([74]) of the inputs $\vec{x}_i$. Suppose that $\mathrm{M} = \sum_i \alpha_i \vec{x}_i \vec{x}_i^\top$, where the coefficients α_i must be learned from examples. An SDP for optimizing these coefficients is obtained by substituting this decomposition of the Mahalanobis distance metric into the semidefinite programming problem. The resulting SDP involves the inputs only through their inner products $\vec{x}_i^\top \vec{x}_j$; if these are computed by a kernel function $k(\vec{x}_i, \vec{x}_j)$, then we obtain a kernelized version of the algorithm.

Strengths:

The MMC algorithm is a good transformation of the input space for k-means clustering. It was one of the first algorithms that learned a Mahalanobis metric with convex optimization.

Limitations:

As the Mahalanobis Metric for Clustering algorithm follows the same intuition as linear discriminant analysis (Section 2.2.1), it also inherits some of its limitations. MMC does

not work if the input data is sampled from multimodal distributions, as it collapses similarly labeled inputs irrespective of the underlying data structure. The lack of a closed form solution to the convex optimization problem makes the algorithm significantly slower than the methods described in Section 2.2.1.

Relationship with LMNN:

MMC is probably the algorithm most similar to LMNN. In fact, LMNN was heavily inspired by this work. Both algorithms are based on semidefinite programming, but LMNN differs in its focus on local neighborhoods for kNN classification. By enforcing the margin to keep impostors out of local neighborhoods, LMNN is much more directly geared towards kNN-based classification. Also, MMC cannot be applied to multi-modal data.

Online learning of Mahalanobis distances

An approach similar to MMC has been introduced by [74]. Their algorithm, named *POLA* - Pseudo-metric Online Learning Algorithm - solves a convex optimization to learn a Mahalanobis metric, but differs from MMC by two main factors: POLA is an online algorithm and it creates a margin which separates similarly- from differently labeled inputs. As with LDA and the convex Mahalanobis metric learning for clustering, the authors follow the intuition of moving differently labeled inputs apart and similarly labeled inputs together. In addition to the matrix $\mathbf{M}$, POLA also learns an adaptive margin threshold b. Similarly labeled inputs should be *at most* a distance of $b-1$ apart. Differently labeled inputs are moved *at least* a distance of $b+1$ apart.

In order to simplify notation, let $(\mathbf{M}_t, b_t) \in \mathcal{R}^{d^2+1}$ be a vector consisting of the column-wise vectorized matrix $\mathbf{M}_t$ and the value for b_t, where $\mathbf{M}_t$ is of size $d \times d$.

The online version of the algorithm receives tuples $(\vec{u}_t, \vec{v}_t, y_t)$ sequentially, where the similarity indicator $y_t = 1$ if and only if the two vectors $\vec{u}, \vec{v}$ are similar, otherwise $y_t = -1$. The algorithm fetches one tuple at a time and updates the Mahalanobis matrix $\mathbf{M}$ and the threshold b. If $\vec{u}_t$ and $\vec{v}_t$ have matching labels, they are moved to be within a squared distance of $b-1$ from each other, otherwise they are forced at least a distance $b+1$ apart.

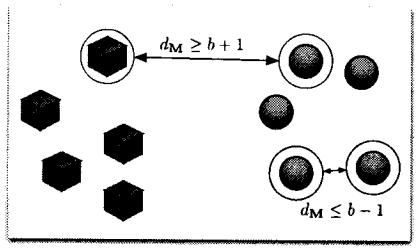


Figure 2.21: The POLA algorithm receives inputs sequentially in pairs. For each pair it updates the Mahalanobis matrix M and the separating margin if necessary.

Letting $\vec{x}_t = \vec{u}_t - \vec{v}_t$, this constraint can be expressed as a single inequality:

$$y_t(b - \vec{x}_t^\top \mathrm{M}\vec{x}_t) \geq 1. \tag{2.28}$$

At each iteration the algorithm does only consider the current tuple $(\vec{u}_t, \vec{v}_t, y_t)$. All past and future tuples are ignored. Figure 2.21 illustrates the intuition behind POLA on a data set where non-matching points are separated by a margin of at least $b + 1$ and matching points are no further than $b - 1$ from each other. Constraint (2.28) can be enforced by projecting onto the convex set of all vectors (M, b) that satisfy it. For each input tuple $(\vec{u}_t, \vec{v}_t, y_t)$ we define:

$$\mathcal{C}_1^t = \{(\mathrm{M}, b) \mid y_t(b - \vec{x}_t^\top \mathrm{M}\vec{x}_t) \geq 1\} \tag{2.29}$$

$$\mathcal{C}_2 = \{(\mathrm{M}, b) \mid \mathrm{M} \succeq 0\} \tag{2.30}$$

At each time-step t, POLA projects the current best matrix M^{t-1} first onto the convex set $\mathcal{C}_1^t$ and then onto $\mathcal{C}_2$.

The online version of POLA can very simply be transformed into a batch algorithm: The input pairs are repeatedly considered until no input pair invokes a hinge loss of more than β, for some constant $\beta > 0$. The number of iterations over the data set can be bounded above, as shown in [74]. Similar to MMC in Section 2.2.2 and LMNN in Section 2.1.8, POLA can also be kernelized (for details see [74]).

Strengths:

Similar to MMC, POLA can greatly improve the performance of clustering algorithms

such as k-means, as long as the data is unimodal. POLA is a very fast online algorithm and can be applied to very large data sets. In addition, it has strong theoretical guarantees regarding the convergence [74].

Limitations:

POLA can generally not be applied to multimodal data as it moves similarly labeled input pairs close together regardless even if they appear in separate, distend clusters. Maintaining the margin between similarly and differently labeled input pairs is not always possible, in which case the theoretical guarantees no longer hold.

Relationship with LMNN:

POLA shares with LMNN that it is based on convex optimization and the separation of non-matching points by a large margin. However, LMNN differs in its focus on local neighborhoods for kNN classification. In particular, LMNN does not seek to minimize the distances between all similarly labeled inputs, only those that are specified as neighbors.

Neighborhood component analysis

Recently, Goldberger et al [35] introduced a novel method for Mahalanobis metric learning, *Neighborhood Component Analysis* (NCA). Just like LMNN, it is especially designed to improve k-nearest neighbor classification. The objective function minimizes the expected leave-one-out classification error of a stochastic variation of kNN classification.

The $k-$nearest neighbor leave-one-out neighbor classification error is a non-continuous, non-differentiable function that would be hard to minimize. Instead, the authors of NCA suggest to use a stochastic neighborhood. For each input $\vec{x}_i$, they define a probability distribution over the remaining inputs to choose the nearest neighbors. The probability of $\vec{x}_j$ to be the nearest neighbor of $\vec{x}_i$ is denoted with the modified softmax

$$p_{ij} = \begin{cases} \frac{\exp\left(-\|\mathbf{A}x_i - \mathbf{A}x_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{A}x_i - \mathbf{A}x_k\|^2\right)} & \text{if } i \neq j \\ 0 & \text{if } i = j, \end{cases} \tag{2.31}$$

where the learned mapping is $\vec{x} \rightarrow \mathbf{A}\vec{x}$. Please note that the scale of $\mathbf{A}$ has a direct influence on the distribution of p_{ij}. NCA not only learns the Mahalanobis distance matrix,

but also indirectly learns the size of the neighborhoods. The stochastic neighborhood assignment has the intriguing advantage over standard kNN that the expected classification error of a given input is a continuous function of $\mathbf{A}$. Let C_{y_i} denotes the set of all inputs of class y_i. The probability that $\vec{x}_i$ is correctly classified can be computed as

$$p_i = \sum_{j \in C_{y_i}} p_{ij}. \tag{2.32}$$

The intuition behind NCA is to maximize the expected number of inputs that are classified correctly. With the help of (2.32), this term can be expressed as follows:

$$f(A) = \sum_i p_i = \sum_i \frac{1}{\sum_{k \neq i} \exp\left(-\|Ax_i - Ax_k\|\right)^2} \sum_{j \in C_i, j \neq i} \exp\left(-\|\mathbf{A}x_i - \mathbf{A}x_j\|^2\right). \tag{2.33}$$

Let us abbreviate $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$. Then we can write the derivative of (2.33) with respect to $\mathbf{A}$ as the following term:

$$\frac{\partial f(\mathbf{A})}{\partial \mathbf{A}} = 2\mathbf{A} \sum_i \left(p_i \sum_k p_{ik} x_{ik} x_{ik}^\top - \sum_{j \in C_i} p_{ij} x_{ij} x_{ij}^\top \right). \tag{2.34}$$

The objective function (2.33) can be maximized with any gradient ascent algorithm. It should be pointed out that different from all previous algorithms in this chapter, this time we do have to deal with local minima issues, as the objective function is not convex. In contrast to the non-convex version of LMNN, NCA does suffer from these local minima and the output heavily depends on the initial starting point. Incidentally, maximizing (2.33) is also equivalent to minimizing the L_1 distance to the perfect probability distribution with uniform non-zero probability assigned over the members of the correct class.

Instead of using a square matrix $\mathbf{A}$, it is also possible to constraint $\mathbf{A}$ to be a rectangular matrix. The resulting Mahalanobis distance then corresponds to the Euclidean distance in a potentially much smaller dimensional vector space. The projection $\vec{x}_i \rightarrow \mathbf{A}\vec{x}_i$ can be used for visualization purposes and to speed up the post-projection kNN classification. Alternatively it is also possible to restrict the matrix $\mathbf{A}$ to be diagonal and learn only a re-scaling of the dimensions. This can speed up the optimization process significantly.

Another obvious extension to NCA is to learn a non-linear function $A(\cdot)$, instead of the matrix $\mathbf{A}$, with the help of a multi-layered neural network. This could potentially lead to even more powerful mappings.

An extension to NCA is Metric Learning by Collapsing Classes (MLCC) by [34]. MLCC is based on the same probabilistic framework as NCA, however can be phrased as a convex optimization problem. In comparison to LMNN, MLCC minimizes the distances to all neighboring points and does not induce a margin.

Strengths: NCA is explicitly geared towards kNN classification and can improve the classification error significantly. It can also be used for supervised dimensionality reduction.

Limitations: The biggest and most obvious drawback of Neighborhood Component Analysis is the non-convex optimization problem. Further, the gradient computation is computationally very involved as it scales $O(n^2d^2)$.

Relationship with LMNN: Both LMNN and NCA are algorithms that learn a linear transformation of the input space to improve nearest neighbor classification. LMNN shares essentially the same goals as NCA, but differs in its construction of a convex objective function. LMNN optimizes a purely energy based loss function in comparison to the probabilistic framework used by NCA. The optimization of NCA is significantly more expensive than LMNN because the gradient has to be re-computed from scratch after every step (in contrast, LMNN allows very cheap gradient updates - for details see Section 2.1.7).

2.2.3 Discussion

In this chapter, we have introduced the large margin nearest neighbor (LMNN) classifier. LMNN learns a Mahalanobis distance metric for kNN classification by semidefinite programming. Our framework makes no assumptions about the structure or distribution of the data and scales naturally to large number of classes. We demonstrated on multiple data sets that the kNN classification improves significantly under the learned metric. Further, we showed that an alternative energy-based classifier almost always leads to better results

than the kNN classifier.

We also investigated several extensions to the to the LMNN algorithm. These included multiple consecutive applications of LMNN, the simultaneous learning of multiple locally linear metrics and a simple kernelization. In addition, we also highlighted the symbiotic relationship between LMNN and tree-based data structures for fast kNN search. In the last section we describe a very simple solver that can solve large scale problems efficiently.

Since its first publication [88], large margin nearest neighbor classification has evolved into a mature and robust classifier based on distance metric learning. We hope that many people will take advantage of the publicly available code base and use LMNN for classification or as a preprocessing tool for further metric based machine learning algorithms.

As the improved performance with multiple metrics suggests, large margin nearest neighbor classification could possibly be extended to more powerful non-linear mappings in the future. Further, it would be interesting to apply the algorithm in a semi-supervised transductive setting, in which only very few labeled inputs are available at training time, but the user is allowed to use the unlabeled test set.

For many applications, such as speech or face identification, the interesting data sets can be orders of magnitudes larger than the MNIST or the 20 newsgroups data set used in this thesis. It is an open challenge to scale large margin nearest neighbor classification up to these kind of data set sizes.

Chapter 3

Unsupervised Metric Learning

In this chapter, we discuss metric learning in the unsupervised scenario. Here, no labels or side information are available. Instead, we will assume that the (potentially high dimensional) input data has intrinsic structure and, in particular, is sampled from an underlying lower dimensional manifold. A manifold is a mathematical surface that behaves locally linear. Representing such data in its raw -high dimensional- form makes dealing with it unnecessarily hard. Euclidean distances are only meaningful on a very local scale, it is very hard to visualize the data and many machine learning algorithms scale at least linear with the input dimensionality. Ideally one would like to have a representation that matches the intrinsic dimensionality of the data and where Euclidean distances are globally meaningful.

Figure 3.1 illustrates an example of a data set sampled from a Swiss roll manifold. The representation on the left is unnecessarily three dimensional. The data has only two true intrinsic degrees of freedom. Further, according to Euclidean distances in the left image, point A is closer to point C than it is to point B. This does not represent the geodesic distance along the manifold, where point A is closer to B than to C. The right image shows a representation where Euclidean distances agree more with the geodesic distances along the manifold.

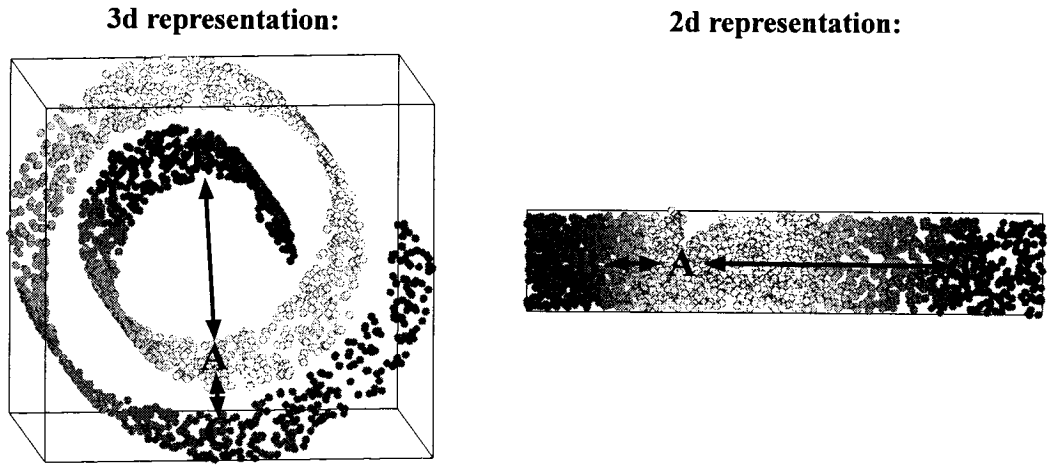


Figure 3.1: Data points sampled from a Swiss roll manifold. The left representation of the data makes point B appear closer to A than C. The right representation reverses the order and agrees with the geodesic distance along the underlying manifold.

3.1 Maximum Variance Unfolding

The following chapter will introduce an algorithm that we refer to as *Maximum Variance Unfolding* (MVU). MVU learns an embedding of the input vectors into a low dimensional feature space. In the feature space, Euclidean distances reflect a better sense of dissimilarity. As in the previous section, the low dimensional embedding is learned via semidefinite programming. However, in contrast to LMNN, MVU does not learn a mapping from the input vectors to feature vectors but learns the output coordinates directly.

3.1.1 Dimensionality Reduction

In the case of high dimensional data, with fewer intrinsic degrees of freedom, it is usually desired to map the input into a lower dimensional Euclidean vector space. This process is generally referred to as *dimensionality reduction.* Given n high dimensional inputs $\vec{x}_i \in \mathcal{R}^p$ where $i \in \{1, 2, \ldots, n\}$, how can we compute outputs $\vec{y}_i \in \mathcal{R}^d$ in one-to-one correspondence with the inputs that provide a faithful representation in $d < p$ dimensions?

By "faithful", we mean that nearby points remain nearby and that distant points remain distant; we shall make this intuition more precise in what follows. Ideally, an unsupervised learning algorithm should also estimate the intrinsic dimensionality d of the manifold sampled by the inputs $\vec{x}_i$.

Our algorithm for manifold learning builds on classical methods for dimensionality reduction [44, 22]. We therefore begin by briefly reviewing the linear methods of principal component analysis (PCA) and metric multidimensional scaling (MDS). The generalization from subspaces to manifolds is then made by introducing the idea of local isometry.

3.1.2 Linear Methods

One of the elementary cornerstones of linear dimensionality reduction is the Johnson Lindenstrauss Lemma [43]. The lemma states that any n points in a high dimensional Euclidean space can be mapped down into an $O(\frac{\log(n)}{\epsilon^2})$ dimensional Euclidean space without distorting the distance between any two points by more than $\pm(1+\epsilon)$. In fact, this mapping can be achieved through random projections. We will briefly give an intuition behind the theorem and refer the interested reader to an excellent proof by Dasgupta and Gupta [24].

An easy way to understand the Johnson Lindenstrauss Lemma is to turn it upside down: Instead of projecting a fixed set of points into a random sub-space, we can also (without loss of generality) project a set of random points onto a fixed sub-space. Imagine we have a random vector $\vec{x} \in \Re^d$ sampled uniformly from the unit hyper-sphere, ie $\|\vec{x}\|^2 = 1$. Let $\vec{x}_p$ be the projection of $\vec{x}$ onto its first k dimensions. It is straight-forward, that the projected point has an expected length of $E(\|\vec{x}_p\|^2) = \frac{k}{d}\|\vec{x}\| = \frac{k}{d}$. (See Figure 3.2 for an illustration of the case with $d = 3$ and $k = 2$.)

Let us further define the mapping $f_k : \vec{x} \rightarrow \sqrt{\frac{d}{k}}\vec{x}_p$. It follows that the expected length of a mapped input is 1:

$$\mu = E(\|f_k(\vec{x})\|^2) = \frac{d}{k}E(\|\vec{x}_p\|^2) = 1. \tag{3.1}$$

In other words, the function f maps the input from its original d dimensions onto a $k \leq d$

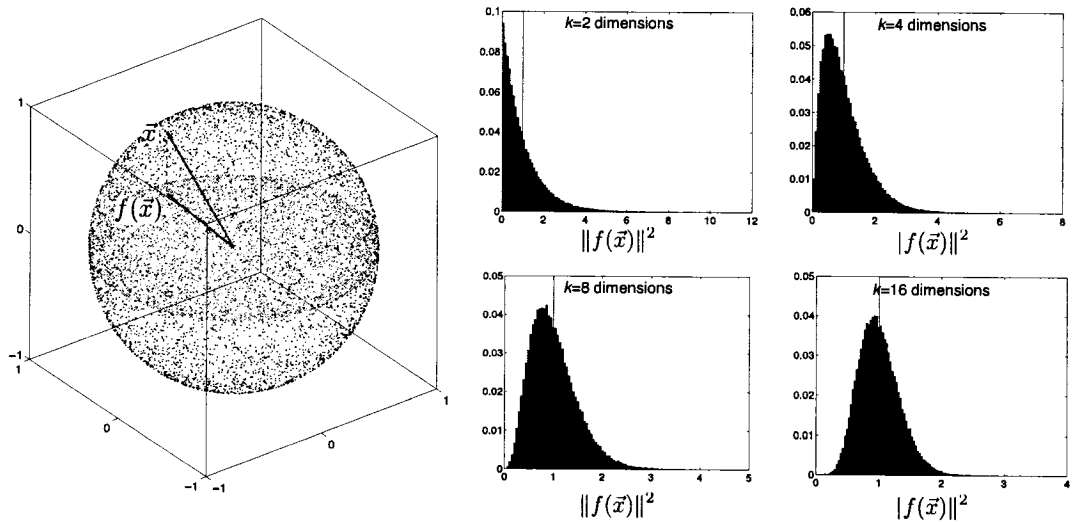


Figure 3.2: *left:* The projection of uniformly sampled points from the unit sphere onto a 2d subspace. The blue line highlights one sample vector. *right:* Histograms of the norm of $n = 100000$ unit vectors in $\Re^{100}$ after the mapping f onto $k = 2, 4, 8, 16$ dimensions.

dimensional sub-space while preserving the distances in expectancy.

Figure 3.2 illustrates the projection the case when $d = 3$ and $k = 2$. The image on the left shows points before (black) and after they are projected (red). The histograms on the right show the empirical distribution of $\|f(\vec{x})\|$ from $n = 10000$ points in a $d = 100$ dimensional Euclidean space. It can be observed that with increasing k the distribution becomes more concentrated around the expected value $\mu = 1$.

The theorem is proven by showing that $\|f(\vec{x})\|^2$, in fact, has to be very tightly concentrated around μ. The probability of deviating by more than a factor of $\pm(1-\epsilon)$ is less than $\frac{1}{n^2}$, with a carefully chosen k to be $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1} \log(n)$. Here, we will skip the formal proof, which is loosely based on applying the Markov inequality to obtain both bounds. For details, please see [24].

From the Johnson Lindenstrauss Lemma we can conclude that random projections into a low dimensional space can preserve distances surprisingly well. It is only natural to ask what the ideal methods for dimensionality reduction are, given that we would like to preserve distances.

miles	NY	LA	Phoenix	Chicago
NY	0	2790	2450	810
LA	2790	0	390	2050
Phoenix	2450	390	0	1740
Chicago	810	2050	1740	0

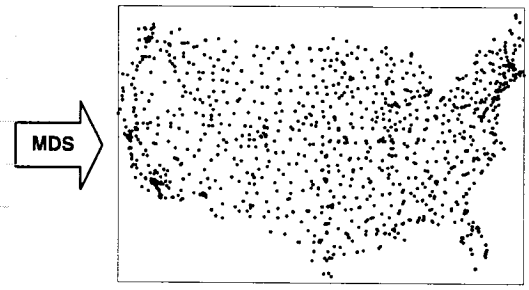


Figure 3.3: Multidimensional scaling applied to a flight-distance matrix of $n = 1097$ US cities. The first five entries of the input matrix are shown on the left. The figure on the right shows the first two dimensions of the resulting embedding.

Multi Dimensional Scaling

One such family of algorithms is multi dimensional scaling (MDS) [22]. MDS focusses on the scenario where we are trying to find a projection which best preserves all distances under an L_2 loss. More precisely, given input vectors $\vec{x}, \ldots, \vec{x}_n \in \Re^d$ and let $\mathbf{D}_{ij} = \|\vec{x}_i - \vec{x}_j\|^2$ we want to find a low dimensional representation $\vec{y}_1 \ldots, \vec{y}_n \in \Re_r$ with $r \ll d$, that minimizes the following loss (or stress-) function:

$$S(\vec{y}_1 \ldots, \vec{y}_n) = \sum_{ij} \left[\mathbf{D}_{ij} - \|\vec{y}_i - \vec{y}_j\|_2^2\right]^2. \tag{3.2}$$

Minimizing the loss function (3.2) is also referred to as *metric*-MDS. Unfortunately, the loss (3.2) suffers from local minima and is non-trivial to minimize. One common relaxation of (3.2) is to preserve inner-products instead of pairwise distances. Let us define the inner-product matrix $\mathbf{G}_{ij} = \vec{x}_i^\top \vec{x}_j$, the altered loss function then becomes

$$S'(\vec{y}_1 \ldots, \vec{y}_n) = \sum_{ij} \left[\mathbf{G}_{ij} - \vec{y}_i^\top \vec{y}_j\right]^2. \tag{3.3}$$

(Here and in what follows, we assume without loss of generality that the inputs are centered on the origin: $\sum_i \vec{x}_i = 0$.) If the loss function in eq. (3.3) is used, the algorithm is generally referred to as *classical* MDS. Figure 3.3 illustrates classical MDS on a data set consisting of all pairwise flight-distances of $n = 1097$ major cities within the continental united states. The inner product matrix was computed from the pairwise distances with

help of the centering matrix $\mathbf{H} = \mathbf{I} - \frac{1}{n}\vec{e}\vec{e}^{\top}$ (where $\vec{e}$ is the all-ones vector). The inner-product matrix is then $\mathbf{G} = -\frac{1}{2}\mathbf{HDH}$. One nice aspect of classical-MDS is that the global minima of S' can be computed efficiently as the eigenvectors of the matrix $\mathbf{G}$. This is relatively straight-forward to see. Let us define $\mathbf{Y} = [y_1, \ldots, y_n]^{\top}$. We can rewrite eq. (3.3) with respect to $\mathbf{Y}$ as

$$S'(\mathbf{Y}) = \|\mathbf{G} - \mathbf{YY}^{\top}\|_F^2. \tag{3.4}$$

If we take the derivative of (3.4) with respect to $\mathbf{Y}$ we obtain the following equation as condition for an extreme point:

$$\mathbf{YY}^{\top}\mathbf{Y} = \mathbf{GY}. \tag{3.5}$$

Let $\vec{u}_1, \ldots, \vec{u}_n$ be the orthonormal eigenvector of $\mathbf{G}$ with respective eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$. It is easy to see that (3.5) is satisfied when the columns of $\mathbf{Y}$ consist of vectors $\sqrt{\lambda_i}\vec{u}_i$. In particular, eq. (3.4) is minimized if we set $\mathbf{Y} = [\sqrt{\lambda_1}\vec{u}_1, \ldots, \sqrt{\lambda_r}\vec{u}_r]$, with a residual error of $S'(\mathbf{Y}) = \sum_{i=r+1}^{n} \lambda_i^2$.

The eigenvectors of the inner-product matrix $\mathbf{G}$ are identical to the input vectors projected onto the eigenvectors of their covariance matrix $C = \frac{1}{n}\sum_i \vec{x}_i\vec{x}_i^T$. In other words, the embedding learned with multidimensional scaling is identical to that of principal component analysis (PCA), as described in Section 2.2.1. Though based on somewhat different geometric intuitions, both PCA and MDS yield essentially a rotation of the inputs followed by a projection into the subspace with the highest variance. The covariance matrix of PCA and the inner product matrix of MDS have the same rank and eigenvalues up to a constant factor. Both matrices are positive semidefinite, and gaps in their eigenvalue spectra indicate that the high dimensional inputs $\vec{x}_i \in \mathcal{R}^p$ lie to a good approximation in a lower dimensional subspace of dimensionality d, where d is the number of appreciably positive eigenvalues. These linear methods for dimensionality reduction generate faithful projections when the inputs are mainly confined to a low dimensional subspace; in this case, their eigenvalues also reveal the correct underlying dimensionality. They do not generally succeed, however, in the case that the inputs lie on a low dimensional manifold.

Non-metric Multi Dimensional Scaling

A non-linear generalization of metric MDS is *non-metric* MDS. In contrast to *metric* MDS, non-metric MDS does not try to match all pairwise distances as close as possible. Instead, the goal is to find an embedding that is true to the relative orderings of the input vectors. The loss function is the Kruskal stress function (also known as stress-1 function):

$$S_1(\vec{y}_1, \ldots, \vec{y}_n) = \min_f \frac{\sum_{ij} \left(f(\|\vec{y}_i - \vec{y}_j\|_2) - \sqrt{\mathbf{D}_{ij}}\right)^2}{\sum_{ij} \mathbf{D}_{ij}}, \tag{3.6}$$

where f is an arbitrary (weakly) monotonic function. The loss (3.6) is generally minimized by iteratively fixing $\vec{y}_i$ while minimizing over f with isotonic regression [3], and fixing f while minimizing over $\vec{y}_i$ with local hill-climbing methods. The optimization is started with some initial "guess" for $\vec{y}_i$. This procedure is continued until convergence.

Unfortunately this optimization requires the dimensionality of $\vec{y}_i$ to be fixed prior to the optimization. Further, it is non-convex and does depend on a suitable initialization for the vectors $\vec{y}_i$. A good choice in practice can be to initialize the vectors $\vec{y}_i$ with the output of classical MDS.

Non-metric MDS does not attempt to preserve all distances directly, and in low dimensions can be more effective than classical MDS. Also, the dissimilarity matrix $\mathbf{D}$ is not required to correspond to squared Euclidean distances. Agarwal et al [67] gave an elementary proof that for any symmetric dissimilarity matrix $\mathbf{D}$, there exists a Euclidean embedding that strictly preserves the ordering of all pairwise dissimilarities. For further details on non-metric MDS, we refer the interested reader to [47].

Local Multi Dimensional Scaling

A more recent extension of multi-dimensional scaling is *local* MDS. Here, the objective function only attempts to preserve local distances [18, 87]. This is particularly helpful if the data set is not in a Euclidean space but instead sampled from a non-linear manifold. In that case only local distances are Euclidean and methods that attempt to preserve all pairwise distances would not be able to discover the low dimensional underlying structure.

The algorithm that we are describing in the remainder of this chapter, which precedes recent work on local-MDS [18, 87], will focus on exactly those kind of data sets that are sampled from underlying manifolds. However, we will not refer to our method as an algorithm for local MDS for mainly historic reasons. Instead, we will emphasize the relationship of our algorithm *maximum variance unfolding* to the more specific family of "manifold learning" algorithms.

3.1.3 From Subspaces to Submanifolds

In the remainder of this chapter, we will focus on data sets that are sampled from underlying Riemannian manifolds. A *manifold* $\mathcal{M}$ is a topological space that is locally Euclidean. A *Riemannian manifold* is a manifold with a *geodesic* metric such that for all $\vec{x}, \vec{z} \in \mathcal{M}$ the distance $dist(\vec{x}, \vec{z})$ is defined as the shortest (geodesic) curve between $\vec{x}$ and $\vec{z}$. Figure 3.4 shows three examples of Riemannian manifolds, each with two intrinsic dimensions embedded inside a three dimensional ambient space. The plot on the right shows a Swiss roll manifold with two points $\vec{x}_i$ and $\vec{x}_j$ and the shortest Euclidean line and geodesic curve between them. It can be observed that the geodesic distance is significantly different from the Euclidean distance. This illustrates that manifolds are only *locally* linear but *not globally*. (Incidentally, the local linearity of the sphere was partly responsible for fooling ancient cultures into believing that the Earth was flat.)

The nonlinear method which we propose in this chapter is based fundamentally on the notion of *isometry*. Formally, two Riemannian manifolds are said to be isometric if there is a diffeomorphism such that the metric on one pulls back to the metric on the other. Informally, an isometry is a smooth invertible mapping that looks locally like a rotation plus translation, thus preserving distances along the manifold. Intuitively, for two dimensional surfaces, the class of isometries includes whatever physical transformations one can perform on a sheet of paper without introducing holes, tears, or self-intersections. Not all manifolds with intrinsic dimensionality d are isometric to a d-dimensional Euclidean

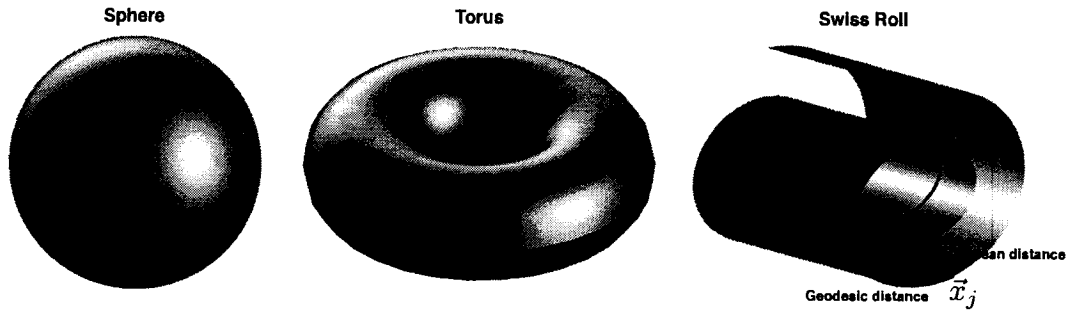


Figure 3.4: Examples of two dimensional manifolds that are embedded inside of a three dimensional ambient space. The manifolds behave locally like a Euclidean space but globally the Euclidean and Geodesic distances can differ substantially (see Swiss roll on the very right).

sub-space. For example, in Figure 3.4 only the Swiss roll manifold is isometric to a connected subspace of the two dimensional plane. However, in practice, many interesting data sets, in particular image manifolds, are isometric to low dimensional connected subsets of Euclidean space [28]. We defer a discussion of competing nonlinear methods based on isometries [29, 82] to section 3.2.

Local isometry

If a data set is sampled from a manifold, it would be desirable to extend the perviously mentioned algorithms, such as PCA or MDS, to preserve this geodesic distance metric instead of the Euclidean. In other words, we would like to design an algorithm that finds an isometric embedding of the input manifold to a Euclidean sub-space. Unfortunately, this is not feasible because in most real world scenarios the underlying manifold and its geodesic metric are unknown. The problem is that we only have the data but not the manifold. Isometry is a relation between manifolds, but we can extend the notion in a natural way to data sets.

Definition 6 *Given two data sets* $\{\vec{x}_i\}_{i=1}^n$ *and* $\{\vec{y}_i\}_{i=1}^n$ *that are in one-to-one correspondence. We say that the data sets are* k**-locally isometric** *if for every point* $\vec{x}_i$*, there exists a rotation and translation that maps* $\vec{x}_i$ *and its* k *nearest neighbors* $\{\vec{x}_{i_1}, \ldots, \vec{x}_{i_k}\}$ *precisely*

onto the points $\vec{y}_i$ *and* $\{\vec{y}_{i_1}, \ldots, \vec{y}_{i_k}\}$.

Here we use the fact that if two vectors $\vec{x}, \vec{z} \in \mathcal{M}$ are sufficiently close then we can use Euclidean distances as an approximation of the Riemannian distance metric, ie $dist(\vec{x}, \vec{z}) \approx \|\vec{x} - \vec{z}\|_2$. With this definition, we can distinguish between linear methods for dimensionality reduction, such as PCA and classical MDS, and the nonlinear method proposed in this chapter. Whereas the linear methods compute mappings that aim to preserve Euclidean distances between all pairs of data points, our method considers the much larger class of nonlinear transformations that only preserve the geometric properties of local neighborhoods. We formalize this approach as an algorithm in the next section.

We will often refer to the outputs $\vec{y}_i$ as providing a low dimensional "embedding" of the high dimensional inputs $\vec{x}_i$. By this, we mean that the outputs provide an explicit low dimensional vector representation of the inputs that faithfully preserves local distances. As in previous work on "manifold learning" [5, 66, 82], it should be emphasized that our algorithm takes as input not the underlying manifold—which may not be known a priori—but a discrete set of high dimensional vectors. From this input, the algorithm constructs a weighted neighborhood graph and produces vector outputs that locally respect the properties of this graph. In the literature on manifold learning, it is common to refer to the outputs as an "embedding" rather than an "immersion", even though the latter better reflects the mathematical usage of these terms for manifolds. In particular, the algorithm does not necessarily output an embedding of the underlying manifold in the sense of the Nash embedding theorems [61].

3.1.4 Maximum Variance Unfolding

Taking as a starting point the notion of local isometry, we can now formulate the problem of manifold learning more precisely. In particular, given n inputs $\vec{x}_i \in \mathcal{R}^d$, can we find n outputs $\vec{y}_i \in \mathcal{R}^r$, where $r < d$, such that the inputs and outputs are k-locally isometric, or at least approximately so? In this section, exploiting the observation that the outputs are determined up to rotation by their inner product matrix $\mathbf{K}_{ij} = \vec{y}_i^\top \vec{y}_j$, we shall show how

to express this problem as a semidefinite program (SDP).

Like PCA and MDS, the algorithm we propose for manifold learning is based on a simple geometric intuition. Imagine each input $\vec{x}_i$ as a steel ball that is connected to its k nearest neighbors by rigid rods. The effect of the rigid rods is to fix the distances and angles between nearest neighbors, no matter what other forces are applied to the inputs. Now imagine that the inputs are pulled apart, maximizing their total variance subject to the constraints imposed by the rigid rods. Fig. 3.5 shows the unraveling effect of this transformation on inputs sampled from a "Swiss roll". The goal of this section is to formalize the steps of this transformation—in particular, the constraints that must be satisfied by the final solution, and the nature of the optimization that must be performed.

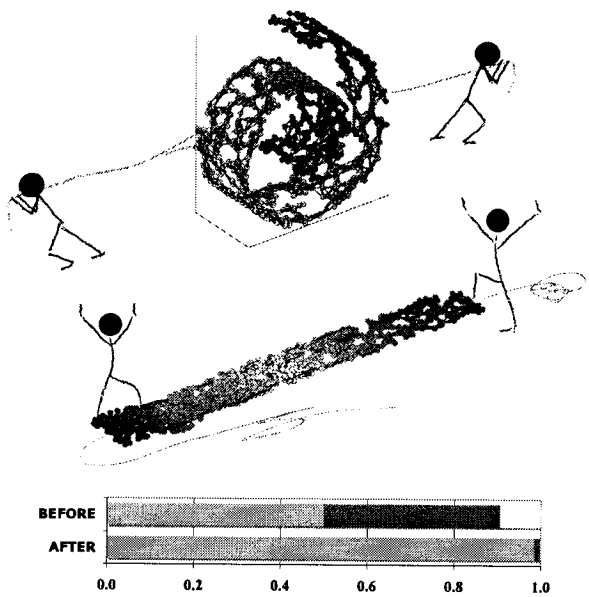


Figure 3.5: Maximum variance unfolding of $n = 800$ data points sampled with noise from a "Swiss roll". The bar diagram below shows the eigenvalues of the inner-product matrices before and after unfolding.

Constraints

The notion of isometry between discrete point sets in section 3.1.3 can be translated into various sets of equality constraints on the inputs $\{\vec{x}_i\}_{i=1}^n$ and the outputs $\{\vec{y}_i\}_{i=1}^n$. To begin, note that neighborhoods of inputs and outputs will be related by translation and rotation if and only if all the distances and angles between points and their neighbors are preserved. Thus, whenever both $\vec{x}_j$ and $\vec{x}_k$ are neighbors of $\vec{x}_i$, for local isometry we must have that:

$$(\vec{y}_i - \vec{y}_j)^\top (\vec{y}_i - \vec{y}_k) \;=\; (\vec{x}_i - \vec{x}_j)^\top (\vec{x}_i - \vec{x}_k)\,. \tag{3.7}$$

Eq. (3.7) is sufficient for local isometry because the triangle formed by any point and its neighbors is determined up to rotation and translation by specifying the lengths of two sides and the angle between them. In fact, such a triangle is similarly determined by specifying the lengths of all its sides. Thus, we can also say that $\{\vec{x}_i\}_{i=1}^n$ and $\{\vec{y}_i\}_{i=1}^n$ are locally isometric if whenever $\vec{x}_i$ and $\vec{x}_j$ are themselves neighbors or common neighbors of another point in the data set, we have:

$$\|\vec{y}_i - \vec{y}_j\|^2 \;=\; \|\vec{x}_i - \vec{x}_j\|^2. \tag{3.8}$$

This is an equivalent characterization of local isometry as eq. (3.7), but expressed only in terms of pairwise distances.

The constraints that we need to impose for local isometry are naturally represented by a graph with n nodes, one for each input. Consider the graph formed by connecting each input to its k nearest neighbors, where k is a free parameter of the algorithm. For simplicity, we assume that the graph formed in this way is connected; if not, then each connected component should be analyzed separately. The constraints for local isometry under this neighborhood relation are simply to preserve the lengths of the edges in this graph, as well as the angles between edges at the same node. In practice, it is simpler to deal only with constraints on distances, as opposed to angles. To this end, we can further connect the graph by adding edges between the neighbors of each node (if they do not already exist). In other words, we create a fully connected clique of size $k+1$ out of

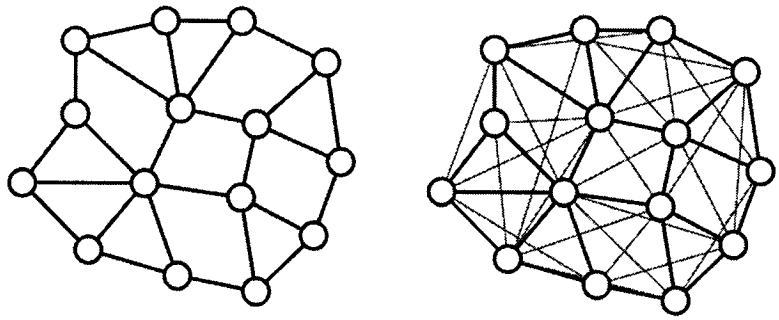


Figure 3.6: A neighborhood graph of $k = 3$ nearest neighbors with (right) and without (left) additional angle preserving edges.

every input $\vec{x}_i$ and its k nearest neighbors. By preserving the distances along edges in this new graph (see Fig. 3.6), we preserve both the distances along edges and the angles between edges in the original graph—because if all sides of a triangle are preserved, so are its angles.

In addition to imposing the constraints represented by the "neighborhood graph", we also constrain the outputs $\vec{y}_i$ to be centered on the origin:

$$\sum_i \vec{y}_i = \vec{0}. \tag{3.9}$$

Eq. (3.9) simply removes a translational degree of freedom from the final solution.

Optimization

What objective function can we optimize to "unfold" a manifold, as in Fig. 3.5? As motivation, consider the ends of a piece of string, or the corners of a flag. Any slack in the string serves to decrease the (Euclidean) distance between its two ends; likewise, any furling of the flag serves to bring its corners closer together. More generally, we observe that any "fold" between two points on a manifold serves to decrease the Euclidean distance between the points. This suggests an optimization that we can perform to compute the outputs $\vec{y}_i$ that unfold a manifold sampled by inputs $\vec{x}_i$. In particular, we propose to

maximize the sum of pairwise squared distances between outputs:

$$\Phi = \frac{1}{2n} \sum_{ij} \|\vec{y}_i - \vec{y}_j\|^2. \tag{3.10}$$

By maximizing eq. (3.10), we pull the outputs as far apart as possible, *subject to the constraints in the previous section.*

We can verify that this objective function is indeed bounded, meaning that we cannot pull the outputs infinitely far apart. Intuitively, the constraints to preserve local distances (and the assumption that the graph is connected) prevent such a divergence. More formally, for any input $\vec{x}_i$, let $\mathcal{N}_i$ be the set of indices of its k nearest neighbors. Let τ be the maximal distance between any two neighbors:

$$\tau = \max_{i,j \in \mathcal{N}_i} \|\vec{x}_i - \vec{x}_j\|^2. \tag{3.11}$$

Assuming the graph is connected, then the longest path through the graph has a distance of at most $n\tau$. We observe furthermore that given two nodes, the distance of the path through the graph provides an upper bound on their Euclidean distance. Thus, for all outputs $\vec{y}_i$ and $\vec{y}_j$, we must have $\|\vec{y}_i - \vec{y}_j\| \leq n\tau$. Using this to provide an upper bound on the objective function in eq. (3.10), we obtain:

$$\Phi \leq \frac{1}{2n} \sum_{ij} (n\tau)^2 = \frac{n^3\tau^2}{2}. \tag{3.12}$$

Thus, the objective function cannot increase without bound if we enforce the constraints to preserve local distances on a connected graph.

Let us now collect the costs and constraints of the optimization to maximize the variance of the outputs $\{\vec{y}_i\}_{i=1}^n$ subject to the constraints that they are centered on the origin and locally isometric to the inputs $\{\vec{x}_i\}_{i=1}^n$. Let $\eta_{ij} \in \{0, 1\}$ indicate whether there is an edge between $\vec{x}_i$ and $\vec{x}_j$ in the graph formed by pairwise connecting all k-nearest neighbors. Then, in terms of the squared distance matrix $\mathbf{D}_{ij} = \|\vec{x}_i - \vec{x}_j\|^2$, the optimization can be written as:

Maximize $\sum_{ij} \|\vec{y}_i - \vec{y}_j\|^2$ **subject to:**
(1) $\sum_i \vec{y}_i = 0$.
(2) $\|\vec{y}_i - \vec{y}_j\|^2 = \mathbf{D}_{ij}$ **for all** (i,j) **with** $\eta_{ij} = 1$.

The optimization as stated above is not convex, as it involves maximizing a quadratic form subject to quadratic equality constraints.

We can obtain a simpler optimization by reformulating the problem in terms of the elements of the inner product matrix, $\mathbf{K}_{ij} = \vec{y}_i^{\mathsf{T}} \vec{y}_j$. As mentioned previously, the matrix $\mathbf{K}_{ij}$ determines the outputs up to rotation. The distance and centering constraints in eqs. (3.8–3.9) are easily expressed in terms of these matrix elements. For example, expanding the square in eq. (3.8) and substituting $\mathbf{K}_{ij} = \vec{y}_i^{\mathsf{T}} \vec{y}_j$, we obtain:

$$\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} = \mathbf{D}_{ij} \tag{3.13}$$

Likewise, the centering constraint in eq. (3.9) can be expressed in terms of these inner products as:

$$0 = \left\| \sum_i \vec{y}_i \right\|^2 = \sum_{ij} \vec{y}_i^{\mathsf{T}} \vec{y}_j = \sum_{ij} \mathbf{K}_{ij}. \tag{3.14}$$

Note that both eqs. (3.13–3.14) are linear equality constraints on the elements of $\mathbf{K}_{ij}$. Writing the constraints in this way, and noting that the outputs are determined up to rotation by their inner products, we may view our original problem as an optimization over inner product matrices $\mathbf{K}_{ij}$ rather than vectors $\vec{y}_i$. Not all matrices, however, can be interpreted as inner product matrices: only symmetric matrices with nonnegative eigenvalues can be interpreted in this way. Thus, we must further constrain the optimization to the cone of symmetric positive semidefinite matrices [85]. We can write this constraint as

$$\mathbf{K} \succeq 0. \tag{3.15}$$

In sum, there are three types of constraints on the inner product matrix $\mathbf{K}_{ij}$, arising from local isometry, centering, and positive semidefiniteness. The first two involve linear equality constraints; the last one is not linear, but importantly it is *convex*. We will exploit

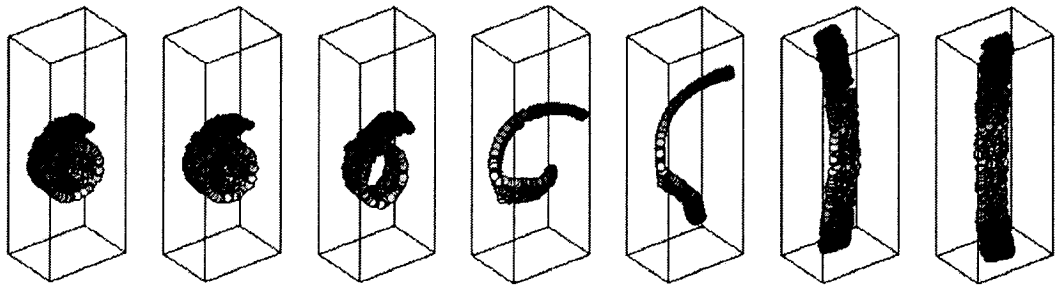


Figure 3.7: Intermediate (non-optimal) solutions to the MVU optimization problem of data points sampled from a Swiss roll.

this property in what follows. Note that there are $O(nk^2)$ constraints on $O(n^2)$ matrix elements, and that the constraints are not incompatible, since at the very least they are satisfied by the original inner product matrix $\mathbf{G}_{ij} = \vec{x}_i^\top \vec{x}_j$ (assuming that the inputs $\vec{x}_i$ are centered on the origin).

Finally, we need to express the objective function in eq. (3.10) directly in terms of the inner product matrix $\mathbf{K}_{ij}$ of the outputs $\vec{y}_i$. Expanding the terms on the right hand side, and enforcing the constraint that the outputs are centered on the origin, we obtain:

$$\begin{aligned}
\Phi &= \frac{1}{2n}\sum_{ij}\left(\|\vec{y}_i\|^2 + \|\vec{y}_j\|^2 + 2\vec{y}_i^\top\vec{y}_j\right), && (3.16)\\
&= \sum_i \|\vec{y}_i\|^2, && (3.17)\\
&= \sum_i \mathbf{K}_{ii}, && (3.18)\\
&= \mathrm{trace}(\mathbf{K}). && (3.19)
\end{aligned}$$

Thus, we can interpret the objective function for the outputs in several ways: as a sum over pairwise distances in eq. (3.10), as a measure of variance in eq. (3.17), or as the trace of their inner product matrix in eq. (3.19). The second interpretation is reminiscent of PCA, but whereas in PCA we compute the linear projection that maximizes variance, here we compute the k-locally isometric embedding. Put another way, the objective function for maximizing variance remains the same; we have merely changed the allowed form of

the dimensionality reduction. We also emphasize that in eq. (3.19), we are maximizing the trace, not minimizing it. While a standard relaxation to minimizing the rank [31] of a semidefinite matrix is to minimize its trace, the intuition here is just the opposite: we will obtain a low dimensional embedding by maximizing the trace of the inner product matrix. Fig. 3.7 illustrates the connection between increasing variance and reducing dimensionality. The images in this figure were obtained from the intermediate solutions of a variance-maximizing optimization subject to centering and local distance constraints.

Collecting the costs and constraints of the above optimization in terms of the inner product matrix $\mathbf{K}_{ij}$, we can write the problem as follows:

Maximize trace(K) subject to:

(1) $\mathbf{K} \succeq 0$.

(2) $\Sigma_{ij}\mathbf{K}_{ij} = 0$.

(3) $\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} = \mathbf{D}_{ij}$ **for all** (i, j) **with** $\eta_{ij} = 1$.

This problem is an instance of semidefinite programming (SDP) [85]: the domain is the cone of positive semidefinite matrices intersected with hyperplanes (represented by equality constraints), and the objective function is linear in the matrix elements. The optimization is bounded above by eq. (3.12); it is also convex, thus eliminating the possibility of spurious local maxima. The problem is guaranteed to be feasible because the constraints are trivially satisfied by the inner product matrix $\mathbf{G}_{ij} = \vec{x}_i^{\top}\vec{x}_j$ of the inputs (assuming that the inputs are centered). There exists a large literature on efficiently solving SDPs, as well as a number of general-purpose toolboxes. The results in this chapter were obtained using the CSDP v4.9 toolbox [14] in MATLAB.

Spectral Decomposition

From the inner product matrix learned by semidefinite programming, we can recover the outputs $\vec{y}_i$ by matrix diagonalization. Let $\mathbf{V}_{\alpha i}$ denote the i^{th} element of the α^{th} eigenvector,

with eigenvalue λ_α. Then the inner product matrix can be written as:

$$\mathbf{K}_{ij} = \sum_{\alpha=1}^{n} \lambda_\alpha \mathbf{V}_{\alpha i} \mathbf{V}_{\alpha j}. \tag{3.20}$$

An n-dimensional embedding that is k-locally isometric to the inputs $\vec{x}_i$ is obtained by identifying the α^{th} element of the output $\vec{y}_i$ as:

$$\mathbf{Y}_{\alpha i} = \sqrt{\lambda_\alpha} \mathbf{V}_{\alpha i}. \tag{3.21}$$

The eigenvalues of K are guaranteed to be nonnegative. Thus, from eq. (3.21), a large gap in the eigenvalue spectrum between the d^{th} and $(d+1)^{\text{th}}$ eigenvalues indicates that the outputs lie in or near a subspace of dimensionality d. In this case, a low dimensional embedding that is *approximately* locally isometric is given by truncating the elements of $\vec{y}_i$. This amounts to projecting the outputs into the subspace of maximal variance, assuming the eigenvalues are sorted from largest to smallest. The quality of the approximation is determined by the size of the truncated eigenvalues; there is no approximation error for zero eigenvalues. The situation is analogous to PCA and MDS, but here the eigenvalue spectrum reflects the dimensionality of an underlying manifold, as opposed to merely a subspace.

The three steps of the algorithm are summarized in Table 3.1. In its simplest formulation, the only free parameter of the algorithm is the number of nearest neighbors in the first step (though one can imagine more elaborate schemes for determining neighborhoods). The second step of the algorithm, involving semidefinite programming, is the most computationally intensive. The first and third steps resemble those of other algorithms for manifold learning, discussed in Section 3.2; our algorithm has rather different properties, however, due to the particular nature of its second step. In earlier work [90, 91], we referred to this algorithm as "semidefinite embedding", but here we will adopt the name "maximum variance unfolding" (MVU), coined by others [79], as it is both more descriptive and less likely to be confused with other work in graph embedding that makes use of semidefinite programming [12, 72].

(I) Nearest Neighbors	Compute k nearest neighbors. Form the graph that connects each input to its neighbors, as well as each neighbor to other neighbors of the same input.
(II) Semidefinite Programming	Compute the inner product matrix of the maximum variance unfolding that is centered on the origin and preserves the distances of all edges in the neighborhood graph.
(III) Spectral Decomposition	Compute a low dimensional embedding from the top eigenvectors of the inner product matrix learned by semidefinite programming.

Table 3.1: The three steps of maximum variance unfolding (MVU), involving nearest neighbor search, semidefinite programming, and matrix diagonalization.

3.1.5 Experimental Results

We evaluated the algorithm in Table 3.1 on several high dimensional data sets whose inputs were either explicitly sampled or believed to have been sampled from a low dimensional manifold.

Fig. 3.5 shows the maximum variance unfolding of $n=800$ inputs sampled from a "Swiss roll". The inputs had $p=8$ dimensions, consisting of the three dimensions shown in the top panel of the figure, plus five extra dimensions filled with low variance Gaussian noise. The middle panel of the figure shows the unfolded Swiss roll computed by semidefinite programming: the solution preserves distances between $k=6$ nearest neighbors. The eigenvalues of the inner product matrix (before and after unfolding) are shown in the bottom panel, normalized by their sum. There are two dominant eigenvalues—a major eigenvalue, representing the unwrapped length of the Swiss roll, and a minor eigenvalue, representing its width. (The unwrapped Swiss roll is much longer than it is wide.) The other eigenvalues are nearly zero, indicating that the algorithm has discovered the correct dimensionality ($d=2$) of the underlying manifold.

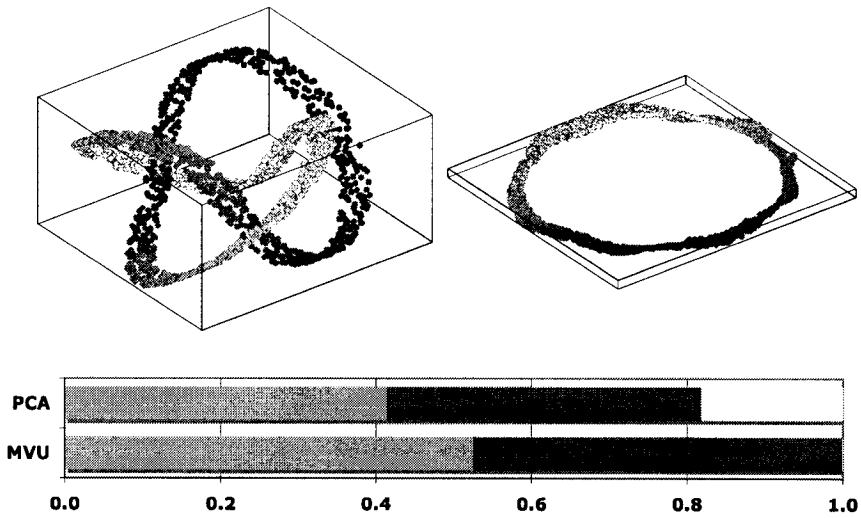


Figure 3.8: 1617 inputs sampled from a trefoil knot in $p=3$ dimensions with corresponding low dimensional embedding computed with MVU.

Fig. 3.8 shows another easily visualized example. The top left panel shows $n=1617$ inputs sampled from a trefoil knot in $p=3$ dimensions; the top right panel shows the maximum variance unfolding that preserves distances between $k=4$ nearest neighbors. The color coding reveals that local neighborhoods have been preserved. The eigenvalue spectrum in the bottom panel reveals two dominant eigenvalues; the rest are essentially zero, again indicating the correct dimensionality ($d=2$) of the underlying manifold—in this case, an annulus.

The bottom section of Fig. 3.9 shows the maximum variance unfolding of color images of a three dimensional solid object. The images were created by viewing a teapot from different angles in the plane. The images have 76×101 pixels, with three byte color depth, giving rise to inputs of $p=23028$ dimensions. Though very high dimensional, the images in this data set are effectively parameterized by one degree of freedom—the angle of rotation. MVU was applied to $n=400$ images spanning 360 degrees of rotation, with $k=4$ nearest neighbors used to generate a connected graph. The two dimensional embedding

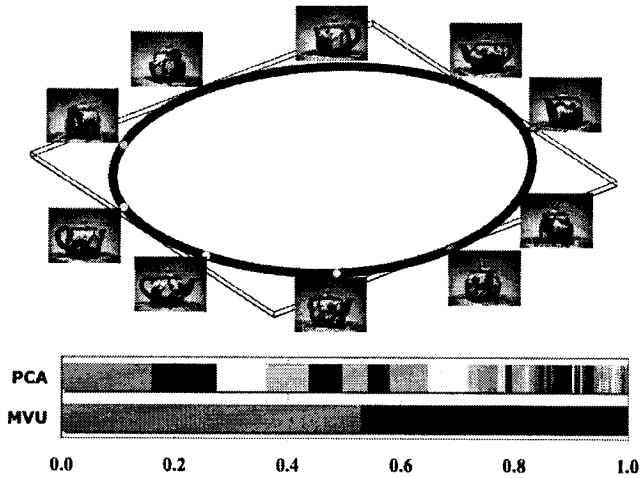


Figure 3.9: *Top:* two dimensional embedding of $n=400$ images of a rotating teapot, computed by MVU with $k = 4$ nearest neighbors. For this experiment, the teapot was rotated 360 degrees; the low dimensional embedding is a full circle. A representative sample of images are superimposed on top of the embedding. *Bottom:* eigenvalues from the matrices in PCA and MVU, shown as a fraction of the trace.

discovered by MVU represents the rotating object as a circle—an intuitive result analogous to the embedding discovered for the trefoil knot. The eigenvalue spectrum of the inner product matrix learned by semidefinite programming is shown in the bottom panel; all but the first two eigenvalues are practically zero, indicating the global dimensionality ($d=2$) of the underlying circle.

Fig. 3.10 was generated from the same data set of images, but using only $n=200$ images that were sampled over 180 degrees of rotation. In this case, the eigenvalue spectrum from MVU detects that the images lie on a one dimensional curve, and the one dimensional embedding orders the images by their angle of rotation.

Fig. 3.11 shows the maximum variance unfolding of another data set of images. In this experiment, the images were a subset of $n=953$ handwritten TWOS from the USPS data set of handwritten digits [40]. The images have 16×16 grayscale pixels, giving rise to inputs with $p=256$ dimensions. Intuitively, one would expect these images to lie on a low dimensional manifold parameterized by such features as size, slant, and line thickness. The top panel of Fig. 3.11 shows the first two dimensions of the embedding computed

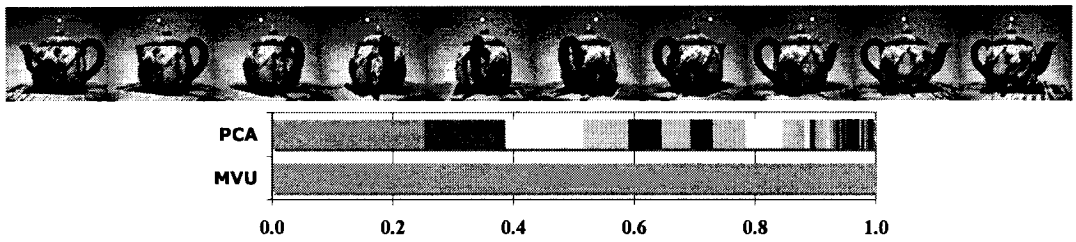


Figure 3.10: *Top:* one dimensional embedding of $n=200$ images of a rotating teapot, computed by MVU with $k = 4$ nearest neighbors. For this experiment, the teapot was only rotated 180 degrees. Representative images are shown ordered by their location in the embedding. *Bottom:* eigenvalues from the matrices of PCA and MVU, shown as a fraction of the trace.

by MVU with $k = 4$ nearest neighbors. The eigenvalue spectrum in the bottom panel indicates a latent dimensionality significantly larger than two, but still much smaller than the number of significant principal components.

Finally, Fig. 3.12 shows the two dimensional embedding of a data set of images of rotating globes. The embedding was computed by MVU with $k = 4$ nearest neighbors. The inputs consisted of $n = 900$ color images at 47×47 pixel resolution, corrupted by white Gaussian noise. The standard deviation of the noise was equal to 30% of the pixel intensity range; for comparison, clean and noisy images are shown to the left and right of the embedding. The viewpoints were evenly sampled over 30 degrees longitude and latitude. The diamond-shaped embedding, with representative (clean) images superimposed, reveals the two degrees of freedom corresponding to longitude and latitude. The eigenvalue spectrum in the bottom panel also indicates the correct dimensionality ($d=2$) of the underlying manifold.

3.1.6 Relaxing the Constraints

It is often desirable to relax the constraints in eq. (3.8) such that local distances are not strictly preserved. In some applications, for example, these distances only provide a rough estimate of proximity relations. Relaxing the constraints always leads to solutions that

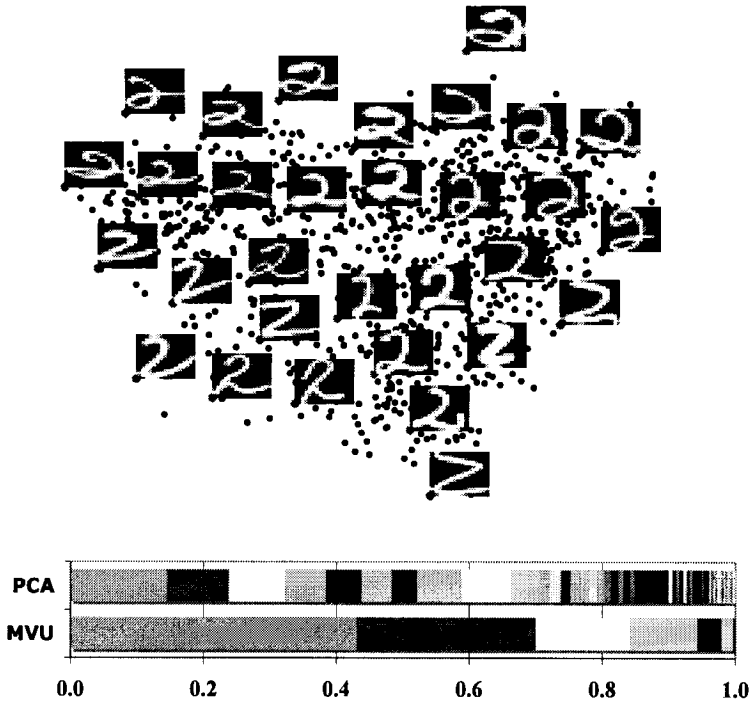


Figure 3.11: *Top:* Maximum variance unfolding of $n = 953$ images of handwritten TWOS with $k = 4$ nearest neighbors. Representative images are shown next to circled points. *Bottom:* eigenvalues from the matrices of PCA and MVU, shown as a fraction of the trace.

have equal or greater variance; the resulting inner product matrices also tend to have fewer numbers of appreciable eigenvalues. Thus, the relaxed optimizations can be viewed as an option for more aggressive forms of dimensionality reduction,

One simple way to relax the constraints is to replace the strict equalities in eq. (3.8) by inequalities. This allows the distances between nearest neighbors to shrink, but not to grow. As the optimization in eq. (3.10) is based on maximizing variance, it acts to minimize the slack in the inequalities even when these constraints are not enforced as strict equalities. The optimization in this case is given by:

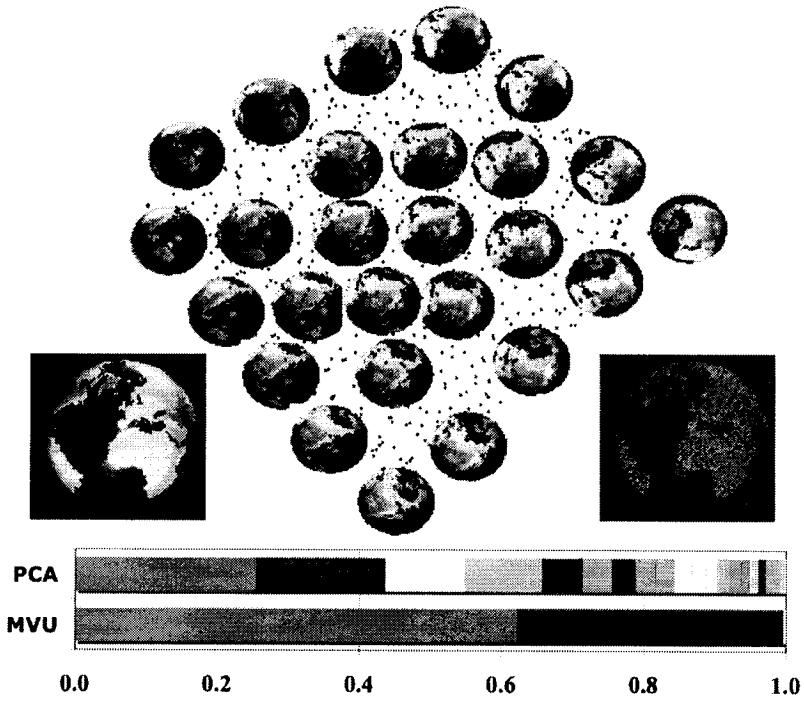


Figure 3.12: The two dimensional embedding of $n=900$ noisy images of a rotating globe, computed by MVU with $k=4$ nearest neighbors.

Maximize trace(K) subject to:

(1) $\mathbf{K} \succeq 0$.

(2) $\Sigma_{ij}\mathbf{K}_{ij} = 0$.

(3) $\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} \leq D_{ij}$ **for all** (i,j) **with** $\eta_{ij}=1$.

Interestingly, the dual of this semidefinite program (with inequalities rather than equalities) arises in the calculation of fastest mixing Markov processes on weighted graphs [79].

Fig. 3.13 shows the maximum variance unfolding of a data set of $n=1960$ images of faces, with inequality constraints from $k = 4$ nearest neighbors. The images contain different views and expressions of the same face. The images have 28×20 grayscale pixels, giving rise to inputs with $p = 560$ dimensions. The top panel in the figure shows a three dimensional embedding of these images. The bottom panel shows the eigenvalue spectra of the matrices from PCA and MVU with both strict equality and relaxed inequality

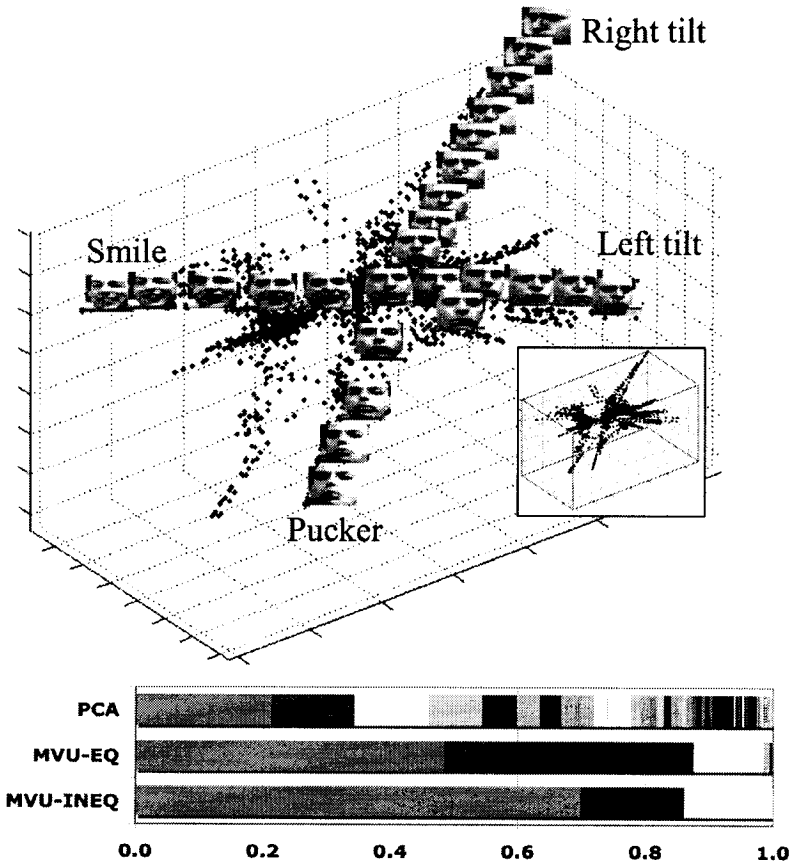


Figure 3.13: Three dimensional embedding of $n = 1960$ images of faces, obtained by MVU using $k=4$ nearest neighbors (and inequality constraints).

constraints. Note how the relaxed optimization concentrates more variance in the leading three dimensions.

Another way to relax the constraints in maximum variance unfolding is to allow the distances to change slightly, but to add a term to the objective function that penalizes this slack. This can be done by introducing slack variables ξ_{ij}, one for each of the constraints in eq. (3.8). The relaxed optimization is given by:

Maximize $(1-\omega)\text{trace}(\mathbf{K}) - \omega \sum_{ij} \eta_{ij} \|\xi_{ij}\|$

subject to:

(1) $\mathbf{K} \succeq 0$.

(2) $\Sigma_{ij} \mathbf{K}_{ij} = 0$.

(3) $\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} = D_{ij} + \xi_{ij}$ **for all** (i,j) **with** $\eta_{ij} = 1$.

The constant $\omega \in (0,1)$ balances the objectives of maximizing variance and penalizing slack. In practice, choosing ω very close to 1 seems to work well for balancing these two objectives.

Fig. 3.14 shows an example where this use of slack allows the algorithm to return an improved result. In this figure, the algorithm was applied with and without slack variables to $n = 800$ inputs sampled *without noise* from a Swiss roll. When distances are strictly preserved between $k = 6$ nearest neighbors, the outputs "lock up" before the data set is completely unfolded; when slack is allowed, however, we obtain the expected result. (Note that the earlier result in Fig. 3.5 was obtained from inputs with five extra dimensions filled with Gaussian noise; the noise makes the problem less rigid.) The main disadvantage of slack variables is the extra computation required to optimize an objective function with $O(nk^2)$ additional variables. The resulting optimization can be noticeably slower.

The above relaxations may prove particularly useful in applications when the distances D_{ij} are not computed from Euclidean distances but are specified in some other way. In this case, the optimization with strict distance-preserving constraints may not be feasible. No matter what the distances D_{ij}, the relaxed versions of maximum variance unfolding always have non-empty feasible regions containing the trivial solution $\mathbf{K}_{ij} = 0$. In such problems, the optimization can be used to return the variance-maximizing embedding in Euclidean space that best preserves local distances.

Finally, we mention one other type of relaxation that has proved useful in an application of maximum variance unfolding to language data [13]. One obtains a simpler problem in semidefinite programming, with many fewer constraints, by only preserving distances

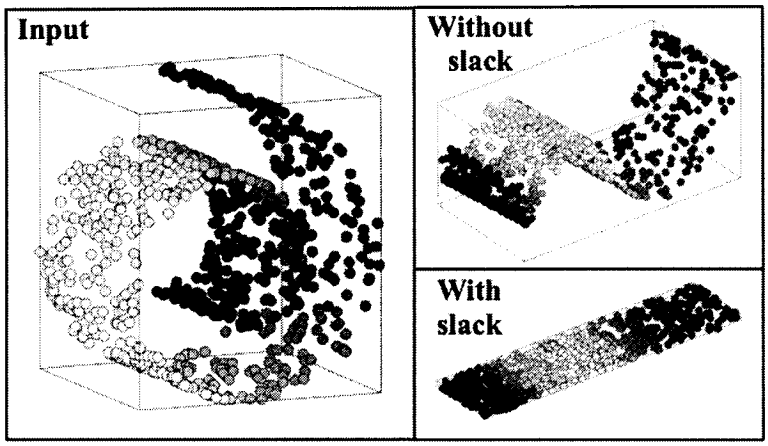


Figure 3.14: Maximum variance unfolding of $n = 800$ inputs sampled without noise from a Swiss roll. Strictly preserving distances between $k = 6$ nearest neighbors causes the outputs to "lock up" before the data set is completely unfolded. Allowing slack in these constraints leads to the desired solution.

to k-nearest neighbors as opposed to preserving distances and angles. The resulting optimization can be used to unfold larger data sets; it also tends to lead to more aggressive forms of dimensionality reduction.

3.1.7 Limits of Manifold Learning

Is the MVU embedding always of lower dimensionality than the input data? Unfortunately, this is not guaranteed. Figure 3.15 shows a synthetic example of $n = 300$ data points sampled from a two dimensional data structure that resembles a comb. The top-left image in Figure 3.15 shows the input data. The underlying structure is locally mostly one-dimensional, except at the points where two lines meet. The bottom images show the first three dimensions of the resulting MVU and Isomap embeddings (obtained with $k = 3, 5$ respectively). The individual strings are pointing away from each other in different directions, therefore maximizing their pairwise distances. The eigenvalues of the respective inner-product matrices (in the bottom of the figure) reveal that the initial two dimensional input was embedded in a vector space with six dimensions of non-zero variance with MVU

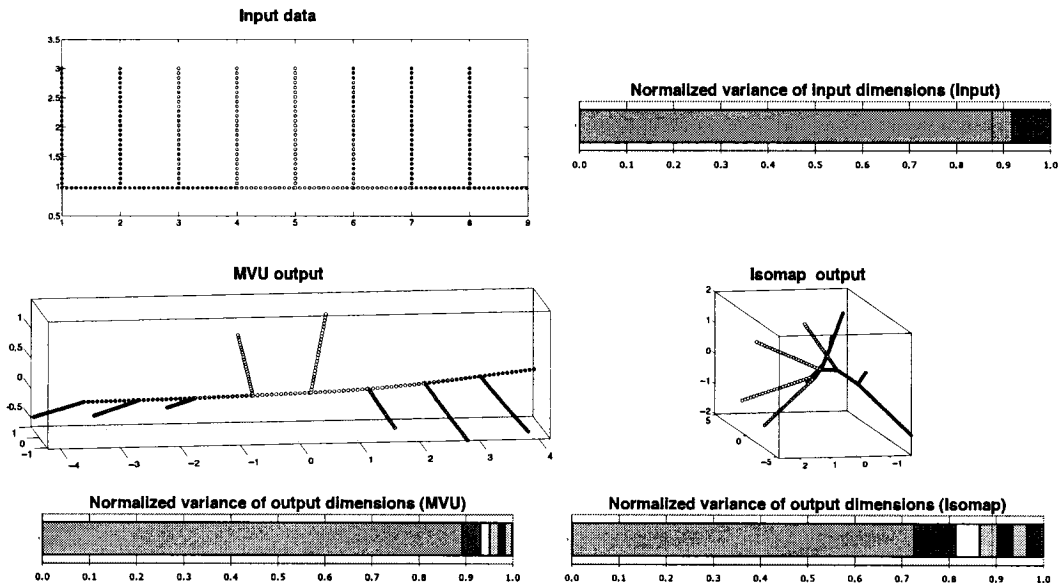


Figure 3.15: An example data set on which MVU and Isomap *increase* the dimensionality of the data.

and eight dimensions with Isomap.

The problem with the "comb" data arises because its intrinsic dimensionality is not well defined. Locally, it is 1-dimensional almost everywhere, but globally it is not isometric to a 1-dimensional subspace. If the neighborhood size would be increased from $k = 3$ to be large enough to span neighborhoods across different strings, the local neighborhoods would have dimensionality 2 and the output becomes identical to the input. It is unclear, however, what the "correct" embedding of this data set is.

3.1.8 Large Scale Extensions

So far, in this chapter, we have introduced maximum variance unfolding, an algorithm for manifold learning. In the previous sections, we demonstrated that MVU can be written as a semidefinite program, which can be solved with off-the-shelf SDP solvers. For example one of the most efficient SDP solvers is CSDP 5.0 [14]. It can solve the SDP from Section 3.1.4 with $n = 1000$ inputs in about 6 minutes, $n = 2000$ in 45 and problems

with $n = 2500$ inputs in around 84 minutes (on a standard 2.4Ghz Intel core duo desktop computer). For many applications, this might not be fast enough. One such application is sensor network localization [11].

For these applications we would like to have a much faster version of MVU. But what is the bottle-neck that makes solving SDPs slow? Most SDP solvers are based on interior-point methods whose time-complexity scales $O(n^3 + c^3)$ with a matrix of size $n \times n$ and c constraints [14]. To be able to apply MVU on larger data sets, even approximately, we must therefore reduce them to SDPs over small matrices with small numbers of constraints.

In this section, we will propose a novel approach for solving large-scale semidefinite programs that occur in the context of maximum variance unfolding. Our approach makes use of a matrix factorization computed from the bottom eigenvectors of the graph Laplacian. The factorization yields accurate approximate solutions which can be further refined by local search. The power of this method will be illustrated by simulated results on sensor localization.

Relaxing Constraints

The first step to scale up the optimization is to reduce the number of constraints. The original SDP from Section 3.1.4 has $O(kn)$ constraints. We showed that in many cases it is convenient to relax the constraints with slack variables, for example in the presence of noise. The relaxed constraint to preserve the distance $\mathbf{D}_{ij}$ is:

$$\forall i, j \;\; \eta_{ij} \left[\mathbf{K}_{ii} + \mathbf{K}_{jj} - 2\mathbf{K}_{ij}\right] = \eta_{ij} \left[\mathbf{D}_{ij} + \xi_{ij}\right]. \tag{3.22}$$

Each pair of indices i, j with $\eta_{ij} = 1$, has its own slack variable ξ_{ij} which absorbs the constraint violation. The objective minimizes the sum of all absolute violations $\sum_{ij} |\xi_{ij}|$. In practice, the absolute value is encoded with two constraint slack variables $\eta_{ij}^{+} > 0$ and $\eta_{ij}^{-} < 0$.

In order to reduce the number of constraints, we would like to collapse them into a single constraint. This is possible, if we square the penalty. (Not squaring the penalty would

result in a much more complicated SDP.) Instead of two slack variables per constraint, we will only have one single slack variable ξ which absorbs the sum of all squared violations. The single local-distances constraint becomes:

$$\sum_{ij} \eta_{ij} \left[\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} - \mathbf{D}_{ij}\right]^2 = \xi^2. \tag{3.23}$$

The constraint in eq (3.23) is quadratic in $\mathbf{K}$ and is of the family of second order cone constraints. Second-order cone constraints are less computationally expensive than semidefinite constraints and in this case no bottleneck [56]. We can now substitute eq (3.23) into the semidefinite program:

Maximize $(1-\omega)\text{trace}(\mathbf{K}) - \omega\xi$

subject to:

(1) $\mathbf{K} \succeq 0$.

(2) $\Sigma_{ij}\mathbf{K}_{ij} = 0$.

(3) $\sum_{ij} \eta_{ij} \left[\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} - \mathbf{D}_{ij}\right]^2 \leq \xi$.

This relaxed formulation of the SDP leaves us with only three constraints. One nonlinear convex constraint to ensure that $\mathbf{K}$ is positive semidefinite $(\mathbf{1})$, one linear constraint to make $\mathbf{K}$ centered $(\mathbf{2})$ and one second order cone constraint that enforces the preservation of local distances $(\mathbf{3})$. We have now reduced the number of contains from $O(kn)$ to 3. The next step is to reduce the size of the target matrix.

Matrix factorization

To obtain an optimization involving smaller matrices, we appeal to ideas in spectral graph theory [20]. Recall that the input is presented as a connected graph whose edges represent local pairwise connectivity. Whenever two nodes share an edge in this graph, we want the output locations of these nodes to be relatively similar. We can view the mapping from input to output as a function that is defined over the nodes of this graph. Because the edges represent local distance constraints, we expect this function to vary smoothly

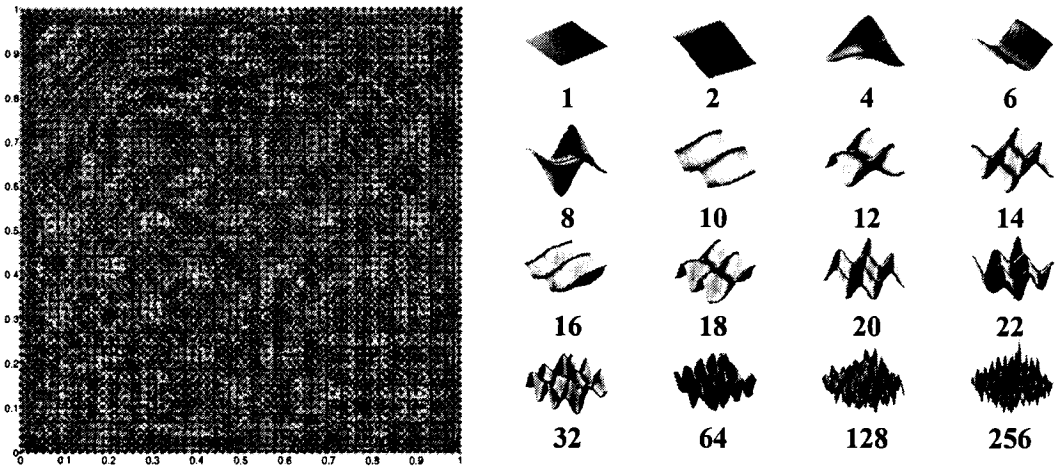


Figure 3.16: A visualization of eigenvectors of the graph Laplacian on a regularly sampled graph inside the unit square. See text for detailed information.

as we traverse edges in the graph. The idea of graph regularization in this context is best understood by analogy. If a smooth function is defined on a bounded interval of $\Re^1$, then from real analysis, we know that it can be well approximated by a low order Fourier series. A similar type of low order approximation exists if a smooth function is defined over the nodes of a graph. This low-order approximation on graphs will enable us to simplify the SDPs for MVU, just as low-order Fourier expansions have been used to regularize many problems in statistical estimation.

Function approximations on graphs are most naturally derived from the eigenvectors of the graph Laplacian [20]. For unweighted graphs, the graph Laplacian L computes the quadratic form

$$f^\top \mathbf{L} f = \sum_{i \sim j} (f_i - f_j)^2 \tag{3.24}$$

on functions $f \in \Re^n$ defined over the nodes of the graph. The eigenvectors of L provide a set of basis functions over the nodes of the graph, ordered by smoothness. Thus, smooth functions f can be well approximated by linear combinations of the bottom eigenvectors of L.

Figure 3.16 shows a visualization of the eigenvectors of a uniformly sampled 16-nearest neighbor graph within the unit square. The graph is shown on the left side of the figure. The right side shows sixteen three dimensional surface plots with the nodes within the unit square as the x, y coordinates and the values of the eigenvectors as the vertical $z-$axis. The small plots are sorted by increasing eigenvalues and each visualizes one eigenvector. The number below the surface plots indicates the position in the eigenvalue ranking (1 corresponds to the smallest eigenvalue). It can be observed, that the surfaces become increasingly less smooth as the eigenvalue increases. (We skipped most odd numbered eigenvectors for symmetry reasons.)

Expanding the output vectors $\vec{y}_i$ in terms of these eigenvectors yields a compact factorization for the inner product matrix $\mathbf{K}$. Suppose that $\vec{y}_i \approx \sum_{\alpha=1}^{m} \mathbf{Q}_{i\alpha}\vec{z}_\alpha$, where the columns of the $n \times m$ rectangular matrix $\mathbf{Q}$ store the m bottom eigenvectors of the graph Laplacian (excluding the uniform eigenvector with zero eigenvalue). Note that in this approximation, the matrix $\mathbf{Q}$ can be cheaply pre-computed from the unweighted input graph, while the vectors $\vec{z}_\alpha$ play the role of unknowns that depend in a complicated way on the local distance estimates $\mathbf{D}_{ij}$. Let $\mathbf{Z}$ denote the $m \times m$ inner product matrix of these vectors, with elements $\mathbf{Z}_{\alpha\beta} = \vec{z}_\alpha \cdot \vec{z}_\beta$. From the low-order approximation to the output locations, we obtain the matrix factorization:

$$\mathbf{K} \approx \mathbf{Q}\mathbf{Z}\mathbf{Q}^\top. \tag{3.25}$$

Eq. (3.25) approximates the inner product matrix $\mathbf{K}$ as the product of much smaller matrices. Using this approximation for large data sets, we can solve an optimization for the much smaller $m \times m$ matrix $\mathbf{Z}$, as opposed to the original $n \times n$ matrix $\mathbf{K}$.

The optimization for the matrix $\mathbf{Z}$ is obtained by substituting eq. (3.25) wherever the matrix $\mathbf{K}$ appears in the SDP from the previous section. Some simplifications occur due to the structure of the matrix $\mathbf{Q}$. Because the columns of $\mathbf{Q}$ store mutually orthogonal eigenvectors, it follows that $\mathrm{tr}(\mathbf{Q}\mathbf{Z}\mathbf{Q}^\top) = \mathrm{tr}(\mathbf{Z})$. Because we do not include the uniform eigenvector in $\mathbf{Q}$, it follows that $\mathbf{Q}\mathbf{Z}\mathbf{Q}^\top$ automatically satisfies the centering constraint, which can therefore be dropped. (In other words, all columns in $\mathbf{Q}$ are orthogonal to the

uniform eigenvector and therefore naturally centered.) Finally, it is sufficient to constrain $\mathbf{Z} \succeq 0$, which implies that $\mathbf{QZQ}^\top \succeq 0$. With these simplifications, we obtain the following optimization:

Maximize $(1-\omega)\text{trace}(\mathbf{Z}) - \omega\xi$

subject to:

(1) $\mathbf{Z} \succeq 0$.

(2) $\sum_{ij} \eta_{ij} \left[[\mathbf{QZQ}^\top]_{ii} - 2[\mathbf{QZQ}^\top]_{ij} + [\mathbf{QZQ}^\top]_{jj} - \mathbf{D}_{ij}\right]^2 \leq \xi$.

Eq. (3.25) can alternately be viewed as a form of regularization, as it constrains neighboring inputs to be mapped to nearby outputs even when the estimated local distances $\mathbf{D}_{ij}$ suggest otherwise (e.g., due to noise). Similar forms of graph regularization have been widely used in semi-supervised learning [17]. We will refer to the output of this optimization problem as *variational MVU*.

Gradient-based improvement

While the matrix factorization in eq. (3.25) leads to much more tractable optimizations, it only provides an approximation to the global minimum of the original optimization in chapter 3.1.4. As suggested in [11], we can refine this approximation by using it as initial starting point for gradient descent in the original non-convex formulation of the relaxed optimization problem from Section 3.1.4. This finds a solution in terms of the output vectors $\vec{y}_i$ directly:

Max $(1-\omega)\sum_{ij} \|\vec{y}_i - \vec{y}_j\|^2 - \omega \sum_{ij} \eta_{ij} \left(\|\vec{y}_i - \vec{y}_j\|^2 - \mathbf{D}_{ij}\right)^2$ **subject to:** $\sum_i \vec{y}_i = 0$.

In general, gradient descent on non-convex functions can converge to undesirable local minima. In this setting, however, the solution of the Laplacian variational SDP provides a highly accurate initialization. Though no theoretical guarantees can be made, in practice we have observed that this initialization often lies in the basin of attraction of the true global minimum.

Our most robust results were obtained by a two-step process. First, starting from the m-dimensional solution of the Laplacian variational SDP from the previous chapter (with matrix factorization), we used conjugate gradient methods[1] to maximize the objective function from the non-convex version. Though this objective function is written in terms of the inner product matrix $\mathbf{K}$, the hill-climbing in this step was performed in terms of the vectors $\vec{y}_i \in \Re^m$. While not always necessary, this first step was mainly helpful for input graphs with irregular (and particularly non-convex) boundaries. It seems generally difficult to represent such boundaries in terms of the bottom eigenvectors of the graph Laplacian. Next, we projected the results of this first step into the $\Re^2$ plane and use conjugate gradient methods to minimize the non-convex optimization problem over $\vec{y}_i$. This second step helps to correct patches of the network where either the graph regularization leads to oversmoothing and/or the rank constraint is not well modeled by MVU. We will refer to the full approximated maximum variance unfolding algorithm with an additional conjugate gradient step as *variational MVU (CG)*.

Laplacian Eigenmaps and Locally Linearly Embedding

The regularization with the eigenvectors of the Graph Laplacian can also be interpreted as a linear transformation of the output of Laplacian Eigenmaps [4]. Naturally, it also makes sense to try similar algorithms for manifold learning, such as Locally Linearly Embedding (LLE) [66]. (For a detailed review of both algorithms see Section 3.2.1.) The LLE matrix can be viewed as a generalization of the Graph Laplacian that also allows negative weights but maintains it to be positive semi-definite.

The Laplacian eigenvectors are known to be a bad representation along the edges of a graph. In the case of, for example, the Swiss Roll, the manifold contains two very long edges on both sides and therefore a large fraction of points are along these edges. It follows that for such long "skinny" data sets, LLE is more suitable for regularization which leads to a "thicker" and more "faithful" two dimensional embedding. Figure 3.17

[1] We are grateful to Dr. Carl Edward Rasmussen for sharing his highly efficient conjugate gradient solver.

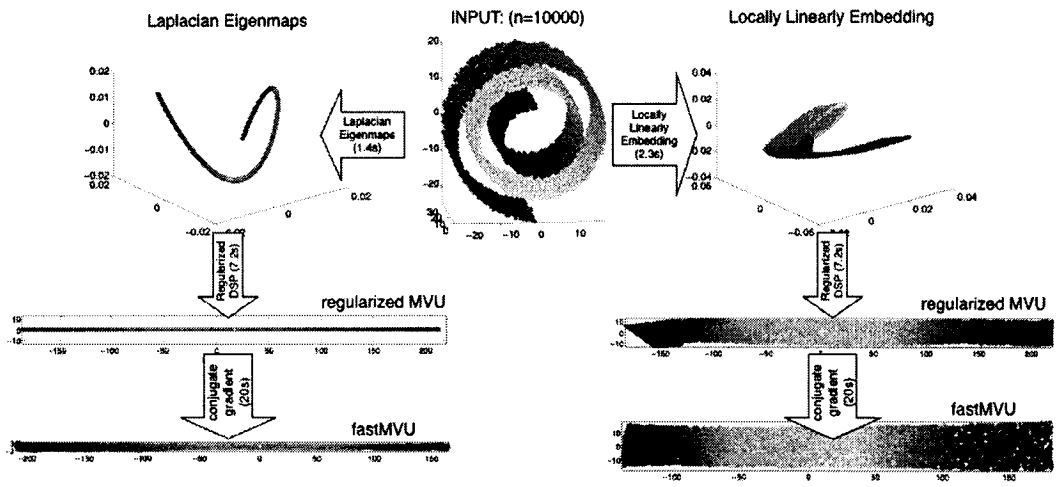


Figure 3.17: The unwrapping of $n = 10000$ data points from a Swiss roll in less than 30 secs. The basis vectors from Laplacian Eigenmaps and LLE lead to different local minima.

shows the comparison of both approaches on a Swiss roll data set.

The downside of the LLE eigenvectors is that the original input vectors are required for the construction of the sparse matrix. In comparison, the computation of the Graph Laplacian only requires a graph with weighted edges. This is a crucial difference for applications such as sensor network localization where the location of the input vectors is unknown (and ultimately to be recovered).

Evaluation

Figure 3.17 shows the schematic layout and the results of variational MVU (CG) with LLE and Laplacian Eigenmaps on a common data set. The $n = 10,000$ input vectors were sampled from a Swiss Roll manifold and corrupted with additive low variance noise. The color coding indicates that local neighborhoods have been preserved. The first three dimensions of the output of Laplacian Eigenmaps and LLE are shown in the top right and top left images, respectively. Both algorithms do not learn a "faithful" low dimensional representation of the input data. These (10 dimensional) representations are then used as

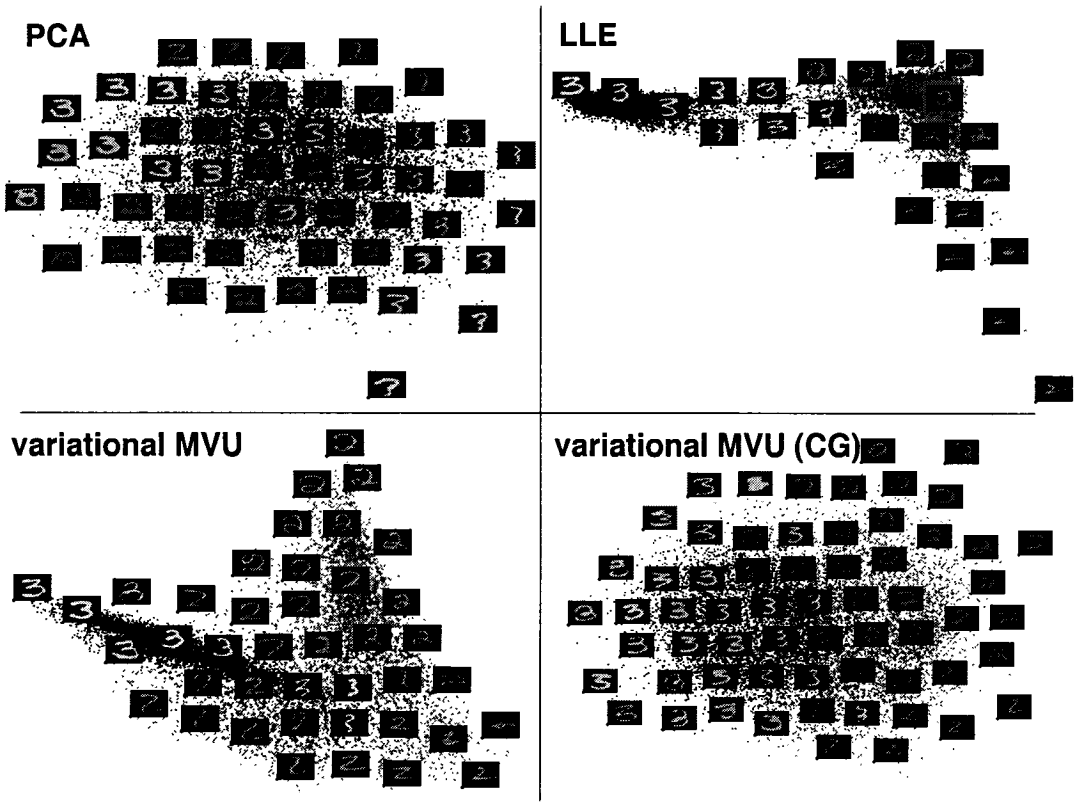


Figure 3.18: Two 2-dimensional embedding of $n = 12089$ handwritten *twos* and *threes*. The four figures show the embeddings obtained with PCA, LLE, and variational MVU with and without conjugate gradient.

basis vectors for the variational MVU. The first two principal components of the respective 10 dimensional outputs are shown in the middle row. In both cases, the variational optimization problems were solved in less than 8 seconds. The final solutions of variational MVU (CG) were obtained after 20 more seconds of conjugate gradient descent (bottom row). In both cases the whole procedure did take no longer than 30 seconds. As mentioned earlier, the eigenvectors of the Laplacian eigenmaps tends to collapse the data points along the edges. The LLE eigenvectors, on the other hand, result in a more "faithful" approximation.

Besides the Swiss roll from Figure 3.17, we also used the mnist handwritten digit data set to learn a low dimensional representation of TWOS and THREES. Figure 3.18 shows the

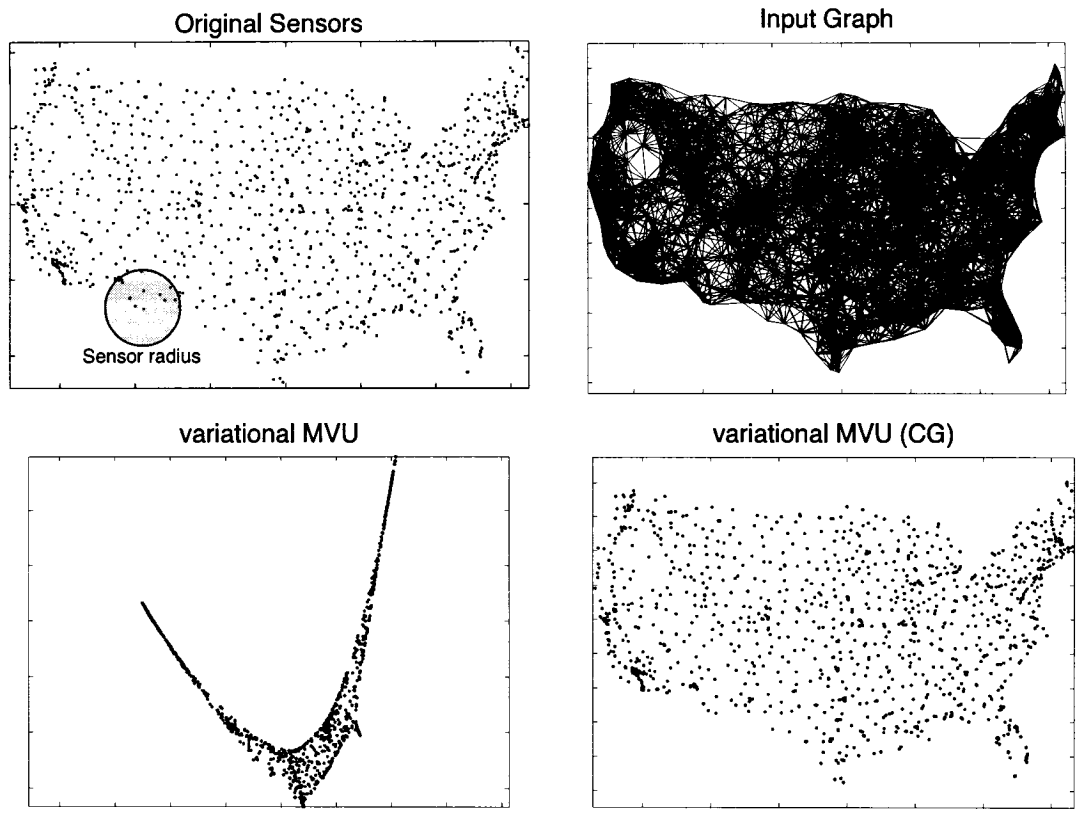


Figure 3.19: Sensor locations inferred for $n = 1097$ largest cities in the continental US.

two dimensional embeddings obtained with PCA, LLE, variational MVU and variational MVU (CG). It can be seen that the first principal component of PCA (horizontal axis) is the direction with maximum variance in ink (top left). The LLE eigenvectors, which take 130s to compute, cluster the vectors mostly into twos and threes (top right). This is not surprising, given that very similar algorithms are used for spectral clustering [62, 77]. variational MVU transforms the $m = 10$ bottom LLE eigenvectors to preserve more local distances in less than 9s (bottom left). After post-processing with conjugate gradient for 170s, the final embedding of variational MVU (CG) is shown on the bottom right.

In addition to manifold data sets, we evaluated our large scale implementation of MVU also on two simulated sensor network data sets of different size and topology. The problem of sensor localization is best illustrated by example; see Figure 3.19. Imagine that

sensors are located in major cities throughout the continental US, and that nearby sensors can estimate their distances to one another, e.g., via radio transmitters. The radio transmitters can only operate locally, and each sensor can obtain a noisy estimate of the distances to other sensors within a small radius. The sensor radius is illustrates in the top left image in Figure 3.19. Given only this local information, the problem of sensor localization is to reconstruct the individual sensor locations and to identify the whole network topology. In purely mathematical terms, the problem can be viewed as computing a low rank embedding $\vec{y}_i$ in two or three dimensional Euclidean space subject to local distance constraints. More explicitly, we are trying to learn vectors $\vec{y}_i$ that minimize the following objective:

$$E(\vec{y}_1, \ldots, \vec{y}_n) = \sum_{i=0}^{n} \eta_{ij} (\|\vec{y}_i - \vec{y}_j\|^2 - \mathbf{D}_{ij})^2 \tag{3.26}$$

where $\eta_{ij} \in \{0, 1\}$ indicates if sensor i and j are within each other's radius and $\mathbf{D}_{ij}$ denotes the (noisy) squared distance between them. Note that the relaxed MVU optimization from the previous section reduces to minimizing (3.26) if $\omega = 1$.

Figure 3.19 shows the original sensors in the top left. The image in the top right displays the input graph generated by connecting each input sensor with all other sensors within its radius. Each edge is weighted by its length which is corrupted by 10% uniform noise. The two images in the bottom show the reconstruction with the variational SDP and after post-processing with conjugate gradient. The variational SDP was solved using the $m = 10$ bottom eigenvectors of the graph Laplacian. From the figure, it can be seen that the solution of the SDP recovers the general topology of the network but tends to distort and clump nodes together, especially near the boundaries. After gradient-based improvement, however, the inferred locations differ very little from the true locations. The construction and solution of the SDP required $3s$ of total computation time on a Intel Core Duo desktop computer, while the post-processing by conjugate gradient descent took an additional $2s$.

The second simulated network, shown in Fig. 3.20, placed nodes at $n = 20000$ uniformly sampled points inside the unit square. The nodes were then centered on the origin. Each node estimated the local distance to up to 20 other nodes within a radius of size $r = 0.06$. The variational SDP was solved using the $m = 10$ bottom eigenvectors of the

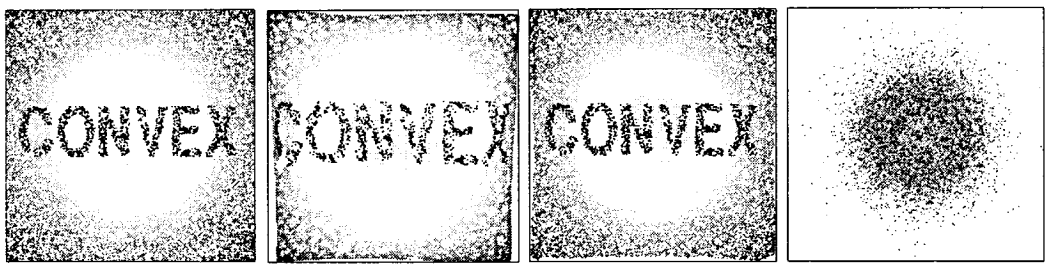


Figure 3.20: Results on a simulated network with $n = 20000$ uniformly distributed nodes inside a centered unit square. See text for details.

graph Laplacian. The computation time to construct and solve the SDP was $19s$. The follow-up conjugate gradient optimization required $52s$ for 100 line searches. Fig. 3.20 illustrates the absolute positional errors of the sensor locations computed in three different ways: the solution from the variational SDP, the refined solution obtain by conjugate gradient descent, and the "baseline" solution obtained by conjugate gradient descent from a random initialization. For these plots, the sensors were colored so that the ground truth positioning reveals the word CONVEX in the foreground with a radial color gradient in the background. The refined solution in the third panel is seen to yield highly accurate results. (Note: the representations in the second and fourth panels were scaled by factors of 0.50 and 1028, respectively, to have the same size as the others.)

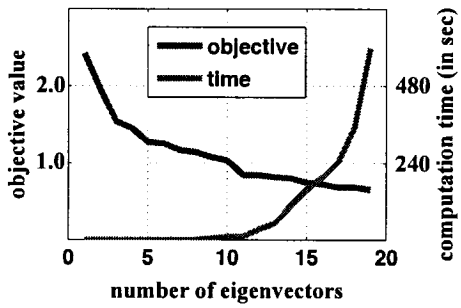


Figure 3.21: *Left:* the value of the reconstruction loss (3.26) after the variational MVU. *Right:* the computation time to solve the SDP. Both are plotted versus the number of eigenvectors, m, in the matrix factorization.

We also evaluated the effect of the number of eigenvectors, m, used in the variational

MVU. (We focused on the role of m, noting that previous studies [11, 57] have thoroughly investigated the role of parameters such as the weight constant ω, the sensor radius r, and the noise level.) For the simulated network with nodes at US cities, Fig. 3.21 plots the value eq. (3.26) as a function of m. Further, it also plots the computation time required to create and solve the SDP. The figure shows that more eigenvectors lead to better solutions, but at the expense of increased computation time. In our experience, there is a "sweet spot" around $m \approx 10$ that best manages this tradeoff. Here, the SDP can typically be solved in seconds while still providing a sufficiently accurate initialization for rapid convergence of subsequent gradient-based methods.

3.1.9 Quantitative Comparison

To compare Maximum Variance Unfolding with previously published work, we need a well-defined quantitative measure of goodness. In the spirit of *precision* and *recall* in information retrieval, we will define two measures: *local continuity* and *local trust*. The first metric measures how much local information of the input is preserved in the low dimensional embedding. The second metric measures how much one can rely that local information in the embedding is also in the input.

As a third measure of goodness, we will define the amount of *local intersection* of neighborhoods within the input and the low dimensional embedding. The definitions of these metrics are heavily inspired by [18] and [87].

Local Continuity

The first metric that we define is *local continuity*. Local continuity measures to what degree the local distances in the input are preserved in the low dimensional output. Let $Y = [\vec{y}_1, \ldots, \vec{y}_n]$ be the matrix containing the output vectors as columns and let $\mathcal{N}(\vec{x}_i)$ be the set of indices of the l nearest neighbors of $\vec{x}_i$. The definition of local continuity is as

follows:

$$C(Y) = 1 - \min_s \frac{\sum_i \sum_{j \in \mathcal{N}(\vec{x}_i)} (s\|\vec{y}_i - \vec{y}_j\| - \|\vec{x}_i - \vec{x}_j\|)^2}{\sum_i \sum_{j \in \mathcal{N}(\vec{x}_i)} \|\vec{x}_i - \vec{x}_j\|^2}. \tag{3.27}$$

Intuitively, $C(Y)$ measures to what degree local distances are preserved in the low dimensional embedding. The scaling factor s is important, because some manifold learning algorithms (for example LLE [66]) learn outputs at a different scale than the input. $C(Y)$ is a measure of goodness, where $C(Y) = 1$ implies a perfect embedding (up to re-scaling by a constant) and $C(Y) = 0$ is the worst case that can trivially be reached for all vectors $\vec{y}_i$ by setting $s = 0$.

Local Trust

Besides preserving local distances, it is also important to measure how much a low dimensional embedding is locally "trustworthy". In other words, we would like to know to what degree local distances in the embedding correspond to local distances in the original input. We can define this measure of *local trust* analogously to (3.27):

$$T(Y) = 1 - \min_s \frac{\sum_i \sum_{j \in \mathcal{N}(\vec{y}_i)} (s\|\vec{x}_i - \vec{x}_j\| - \|\vec{y}_i - \vec{y}_j\|)^2}{\sum_i \sum_{j \in \mathcal{N}(\vec{y}_i)} \|\vec{y}_i - \vec{y}_j\|^2}. \tag{3.28}$$

Similar to eq.(3.27), $T(Y)$ is bounded within the interval $[0, 1]$. A high value of $T(Y)$ means that we can reliably infer local distances in the input spaces from the low dimensional embedding. A low value of $T(Y)$ means that local neighborhoods could have been "intruded" by vectors that were originally further away. This can happen, for example, with linear projections that usually have high continuity, but low trust.

Neighborhood Intersection

As a final measure of goodness, we define *neighborhood intersection.* In contrast to previous metrics, neighborhood intersection does not focus on distance preservation but on the intersection of local neighborhoods. It is defined as follows:

$$N(Y) = \frac{1}{nk} \sum_i |\mathcal{N}(\vec{x}_i) \cap \mathcal{N}(\vec{y}_i)|. \tag{3.29}$$

The measure naturally heavily depends on the definition of the local neighborhoods $\mathcal{N}(\vec{x}_i)$, $\mathcal{N}(\vec{y}_i)$. In the best case, when $\mathcal{N}(\vec{x}_i) = \mathcal{N}(\vec{y}_i)$ for all indices i, we obtain $N(Y) = 1$. Once again, in the worst case, where $\mathcal{N}(\vec{x}_i) \cap \mathcal{N}(\vec{y}_i) = \{\}$ for all i, we obtain $N(Y) = 0$. In our experiments, we defined $\mathcal{N}(\vec{x}_i)$ as the set of the $l = 5$ or $l = 10$ nearest neighbors of $\vec{x}_i$. Please note that as l becomes large $N(Y)$ approaches 1 because trivially all neighbors are preserved if the local neighborhoods span over all input vectors. Hence $N(Y)$ is only an informative metric as long as l is sufficiently small.

Quantitative Experiments

We evaluated various algorithms on four data sets under all three metrics. We used PCA [44] (see Section 2.2.1), hLLE [29], LLE [66], Isomap[82], Laplacian Eigenmaps [4] as described in Section 3.2, MVU and all of its large scale flavors (variational MVU (CG) and variational MVU both after Laplacian Eigenmaps and LLE), as described in Section 3.1.8.

The data sets were chosen to be sufficiently small enough to be able to run all algorithms successfully. The data sets consisted of the Olivetti faces data set (of $n = 400$ images of 10 individuals - see Section 2.1.8), the rotating teapot data set, a synthetic swiss-roll with $n = 2000$ data points and the Frey data set of $n = 1965$ facial expressions, as described in Section 3.1.5. Table 3.2 shows the results of all algorithms on these data sets evaluated under all the metrics. The best result is highlighted in bold. All algorithms were run with a neighborhood graph with $k = 15$ except MVU, which was run on $k = 5$ (because of its higher computational complexity). The output dimensionality was $r = 3$ for all data sets with unknown intrinsic dimensionality and $r = 2$ for the teapot and Swiss roll data set. The output dimensionality was purposely chosen to be either 3 or 2, to focus on an important application of dimensionality reduction: data visualization.

Maximum Variance Unfolding performs surprisingly well according to trust and local intersection, especially keeping in mind that it was trained with a nearest neighbor graph created with $k = 5$ nearest neighbors. (If the local trust of MVU is evaluated with $l =$

statistics	Olivetti Faces	Teapots	Swiss roll	Frey Faces	Average
n	400	400	2000	1965	
k	15	5	15	20	
input dim	1178	23028	3	650	
output dim	3	2	2	3	
local continuity (l=15)					
MVU	0.772	**0.974**	**1.000**	0.813	0.890
var. MVU (CG) (LLE)	**0.964**	0.889	0.956	0.950	0.940
var. MVU (CG) (Lap)	**0.964**	0.973	0.989	**0.952**	**0.970**
var. MVU (LLE)	0.754	0.680	0.958	0.808	0.800
var. MVU (Lap)	0.731	0.940	0.784	0.584	0.760
PCA	0.854	0.702	0.836	0.842	0.809
Isomap	0.850	0.970	0.978	0.834	0.908
hLLE	0.112	0.067	0.865	0.799	0.461
Lap	0.564	0.937	0.659	0.483	0.661
LLE	0.508	0.444	0.844	0.689	0.621
local trust (l=15)					
MVU	0.832	**0.995**	**0.990**	0.716	**0.883**
var. MVU (CG) (LLE)	0.766	0.496	0.600	0.628	0.623
var. MVU (CG) (Lap)	0.778	0.994	0.208	0.629	0.652
var. MVU (LLE)	0.656	0.427	0.672	0.724	0.620
var. MVU (Lap)	0.741	0.922	0.290	0.536	0.622
PCA	**0.839**	0.479	0.389	0.748	0.614
Isomap	0.782	0.993	0.964	0.695	0.858
hLLE	0.047	0.016	0.873	**0.862**	0.449
Lap	0.607	0.921	0.282	0.411	0.555
LLE	0.354	0.329	0.173	0.621	0.369
ocal intersection (l=15)					
MVU	0.491	**0.981**	**0.990**	**0.388**	**0.713**
var. MVU (CG) (LLE)	0.120	0.440	0.824	0.075	0.365
var. MVU (CG) (Lap)	0.131	0.958	0.607	0.112	0.452
var. MVU (LLE)	0.473	0.579	0.828	0.231	0.528
var. MVU (Lap)	**0.511**	0.955	0.665	0.351	0.620
PCA	0.470	0.681	0.397	0.316	0.466
Isomap	0.459	0.979	0.916	0.273	0.657
hLLE	0.369	0.335	0.581	0.008	0.323
Lap	0.473	0.959	0.274	0.343	0.512
LLE	0.406	0.512	0.417	0.169	0.376

Table 3.2: Comparisons of the embeddings of various manifold learning algorithms on four data sets. The table shows the three different metrics that can be used to quantitatively compare the results and their averaged outcomes. (For a description of most of the algorithms, please see Section 3.2.)

5, the average result improves from 0.799 to 0.938.) It has the highest average ranking according to local trust and local intersection.

According to local continuity, it is outperformed by Isomap and both variational MVU (CG) variations. One reason for the underperformance of MVU in local continuity is that MVU was trained with $k = 5$, but the local continuity metric was evaluated on neighborhoods of size $l = 15$. It might be surprising, that variational MVU (CG) (an approximation to MVU) performs better than MVU as far as local continuity is concerned. This can be explained if one takes a look at the difference between variational MVU (CG) and variational MVU on Table 3.2. Keeping in mind that variational MVU (CG) is identical to variational MVU with an additional "tidy up" stage with conjugate gradient, it follows that the conjugate gradient step improves the local continuity metric significantly. This is expected as the conjugate gradient solver operates in the low dimensional space and therefore compensates for the error that was introduced by projecting out the information in non-leading eigenvectors. If the MVU output is used as the initialization for the conjugate gradient solver, similarly good values for local continuity are obtained. Please note that there is no such obvious trend for the other two metrics.

One interesting observation is that MVU obtains almost perfect scores on the Swiss roll and the rotating teapot data set - both data sets that are sampled from true underlying low dimensional manifolds. The Frey facial expressions and the olivetti faces data sets differ in that the data is more than 3 dimensional. In particular, this means that the matrix K has more than three non-zero eigenvalues and by only keeping the $r = 3$ leading eigenvectors, some information may be discarded. Figure 3.22 shows the results of all three metrics evaluated on both faces data sets under varying output dimensionality r. In addition to the estimated local continuity, local trust and local intersection values, Figure 3.22 also displays the normalized eigenvalues. All three metrics improve monotonically with increasing r and plateau as the magnitude of the eigenvalues tends to zero.

Of all non-MVU algorithms, Isomap performs the best on average according to all three metrics. LLE and Laplacian Eigenmaps (abbreviated as Lap) perform relatively

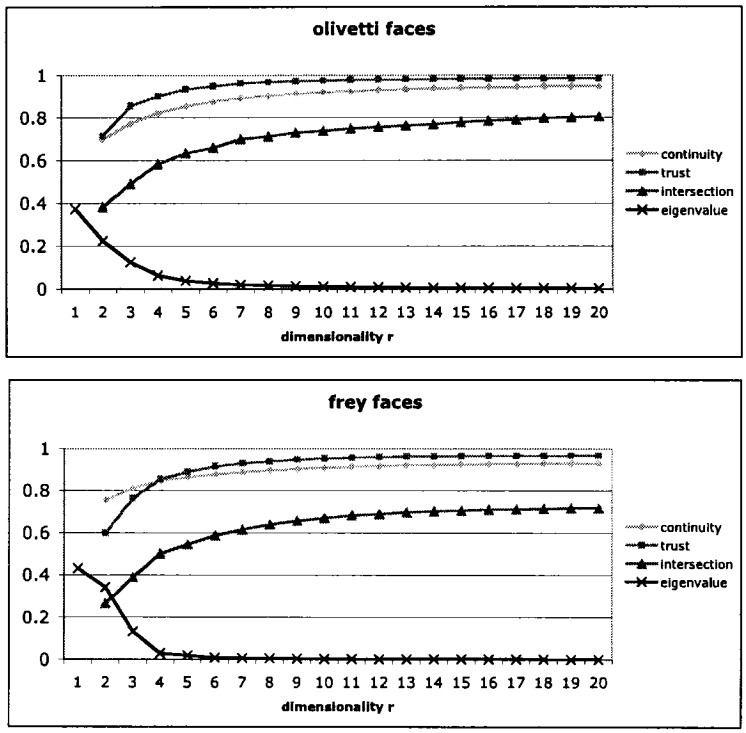


Figure 3.22: The evaluation of the MVU embedding based on local continuity, trust and intersection under varying output dimensionality r. The two data sets are the Frey facial expression data (*top*) and the olivetti faces data set (*bottom*). The eigenvalue shows the variance in each dimension.

poorly but are by far the fastest algorithms. It also needs to be pointed out, that evaluations based on local distance preserving are favoring algorithms based on local isometry such as MVU, Isomap and hLLE. In that context, the relatively bad performance of hLLE is surprising and is most likely due to the known weakness of hLLE to not handle noisy data sets too well. It is only fair to point out that MVU is by far the most computationally demanding of all algorithms.

3.1.10 Out-of-sample Extension

Maximum Variance Unfolding takes a fixed set $\vec{x}_1, \ldots, \vec{x}_n \in \Re^d$ of input vectors and learns a low dimensional representation $\vec{y}_1, \ldots, \vec{y}_n \in \Re^r$ while preserving local distances.

Consequently, this requires that all input vectors $\vec{x}_i$ must be known a-priori - which might not always be the case. In many scenarios, a new input $\vec{x}_t$ might be observed after the embedding of the previous inputs has already been found. Imagine, for example, a face recognition system that analyses images in a low dimensional representation. It is important to be able to take a new, previously unobserved, image and map it into the embedding - without recomputing the entire embedding from scratch. We call this problem *out-of-sample* extension: Given an MVU embedding $\{\vec{y}_i\}_{i=1}^n \in \Re^r$ for some vectors $\{\vec{x}_i\}_{i=1}^n \in \Re^d$, how can we find the corresponding low dimensional representation $\vec{y}_t$ of an out-of-sample vector $\vec{x}_t$?

Inspired by similar problems in sensor network localization [12], and following the spirit of MVU, we will cast the out-of-sample extension problem as a semidefinite program. Assume, without loss of generality, that we reorder the original input data set such that $\vec{x}_1, \ldots, \vec{x}_k$ are the k-nearest neighbors of x_t within $\{\vec{x}_i\}_{i=1}^n$ and $\vec{y}_1, \ldots, \vec{y}_k$ are their respective low dimensional coordinates in $\{\vec{y}_i\}_{i=1}^r$.

Before we can construct our optimization problem we have to remind ourselves of the *Schur Complement Lemma* [15]. In a slightly simplified form, the Schur Complement Lemma states that for any invertible symmetric real matrix $\mathbf{A}$, a symmetric real matrix $\mathbf{C}$ and a real matrix $\mathbf{B}$ the following condition holds:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^\top \\ \mathbf{B} & \mathbf{C} \end{bmatrix} \succeq 0 \Longleftrightarrow \mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^\top \succeq 0, \tag{3.30}$$

where, just as before, "$\succeq 0$" denotes positive semi-definiteness. We can use this lemma to construct a matrix that encodes the low dimensional position of the new input as $\mathbf{B} = \vec{y}_t$. Let us define:

$$\hat{y}_i = \begin{bmatrix} \vec{y}_i \\ -1 \end{bmatrix}, \; \mathbf{S} = \begin{bmatrix} \mathbf{I} & \vec{y}_t \\ \vec{y}_t^\top & s \end{bmatrix}. \tag{3.31}$$

From the definitions in (3.31) it is easy to see that $\hat{y}_i^\top \mathbf{S} \hat{y}_i = \vec{y}_i^\top \vec{y}_i - 2\vec{y}_i^\top \vec{y}_t + s$. We can therefore express the squared distance between $\vec{y}_t$ and any $\vec{y}_i$ in the following way:

$$\|\vec{y}_t - \vec{y}_i\|^2 = \hat{y}_i^\top \mathbf{S} \hat{y}_i + (\vec{y}_t^\top \vec{y}_t - s) \tag{3.32}$$

From equation (3.30) we know that as long as S is positive semi-definite, $s \geq \vec{y}^\top \vec{y}$. If we minimize s, we we will obtain a solution with $s \approx \vec{y}^\top \vec{y}$. We can therefore approximate (3.32) the following way:

$$\|\vec{y}_t - \vec{y}_i\|^2 \approx \hat{y}_i^\top \mathbf{S} \hat{y}_i. \tag{3.33}$$

Let the vector $\vec{e}_i$ be the all-zero vector with a 1 in the i-th position. We will use the vectors $\vec{e}_i$ to access individual elements of S. For example, we can write $s = \vec{e}_{r+1}^\top \mathbf{S} \vec{e}_{r+1}$.

With this notation, we can finally construct our semidefinite programming problem to learn a matrix S as defined in eq (3.31):

Minimize $\sum_i |\xi_i| + \vec{e}_{r+1}^\top \mathbf{S} \vec{e}_{r+1}$ **subject to:**

(1) $\hat{y}_i^\top \mathbf{S} \hat{y}_i = \|\vec{x}_i - \vec{x}_t\|^2 + \xi_i, \;\; i = 1, \ldots, k$

(2) $\vec{e}_i^\top \mathbf{S} \vec{e}_i = 1, \;\; i = 1, \ldots, r$

(3) $\mathbf{S} \succeq 0$

The objective consists of two parts, it minimizes absolute value of the slack variables ξ_i and s. Minimizing s reduces the gap $(s - \vec{y}^\top \vec{y})$ and therefore tightens the first equality in (3.33). Minimizing the magnitude of the slack variables forces the first set of linear constraints to be more strict and tightens the second approximation in (3.33).

The second set of linear constraints ensures that S is of the right form, in particular that the top left corner consists of the identity matrix. The third and last constraint ensures that S is positive semi-definite.

The optimization problem has $2r + 1$ free variables and k linear constraints (ignoring those elements in S that are fixed to the identity matrix). The computational complexity scales with order $O(r^3 + (k+r)^3)$. Considering that both k and r are generally very small, solving the SDP takes about $10ms$ on a standard desktop computer. If $k \geq 2r + 1$ (and if the input points are in general position), the problem has a unique solution. In practice, even for relatively small values of $k (\approx r)$, it is usually the case that $(s - \vec{y}^\top \vec{y})$ is near zero. After training, the vector $\vec{y}_t$ is simply extracted from S.

As an extension, it is also possible to apply conjugate gradient descent from variational

MVU (CG) to the local patch around the inserted vector after the SDP. We evaluate this approach together with just the SDP at the end of this section.

Alternatively to the algorithm proposed in this section, there is also previously published work on out-of-sample extensions for manifold learning [8, 69], which are not as tailored towards the MVU objective. The algorithm by Bengio et al [8] focusses on kernel PCA and Saul and Roweeis [69] follows the LLE objective for local reconstruction. Another approach is to use a non-linear dimensionality reduction algorithm that learn mappings from input to output (but is typically non-convex) such as Gaussian process latent variable model [51] or deep belief nets [68], and initialize it with the embedding obtained from MVU. The learned mapping can then be used for out-of-sample inputs.

Quantitative Evaluation

To quantitatively evaluate the quality of the out-of-sample extension we follow the methodology of Bengio et al, 2004 [8]. Given a data set X, we randomly split it into three disjoint subsets $X = A \cup B \cup C$ where $A \cap B = B \cap C = A \cap C = \{\}$ and $|A| = |B|$. We then computed the low dimensional embeddings of the data set $A \cup C$ and $B \cup C$ with MVU. If MVU is applied to $A \cup C$, let us write Y_A^{AC} to denote the embedding of all elements of A and Y_C^{AC} the embedding of all elements of C.

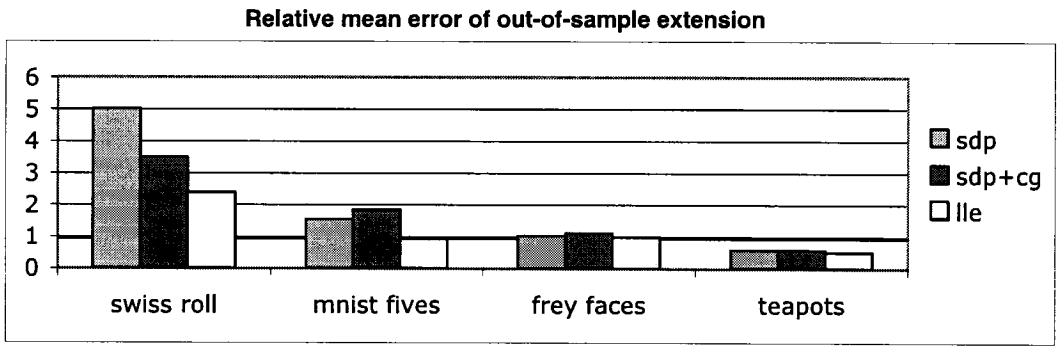


Figure 3.23: The relative regression error of three out-of-sample extension methods. For details see text.

We further applied MVU to A and B individually, to obtain the embeddings Y_A^A and

Y_B^B. We then used the out-of-sample extension algorithm to add all vectors in C to Y_A^A and Y_B^B. We denote the embedding of C obtained with out-of-sample extension on A as $Y_C^{\prime A}$ (and $Y_C^{\prime B}$ respectively).

We evaluate the out-of-sample extensions by a relative measure of consistency:

$$\frac{err(Y_C^{\prime A}, Y_C^{\prime B})}{err(Y_C^{AC}, Y_C^{BC})} \text{ where: } err(X,Y) = \min_Q \|X - QY\|_F. \tag{3.34}$$

In words, the ratio in eq. (3.34) measures how much the out-of-sample extension of set C varies relative to the original algorithm if some parts of the manifold data change (up to a linear transformation Q).

Figure 3.23 shows the (harmonic) mean of the error (3.34) of ten trials on four different data sets. The out-of-sample extensions used are the SDP-based approach from this chapter (denoted "sdp"), the same method with conjugate gradient post-processing ("sdp cg") and the method described in Saul and Roweis [69] ("lle") which positions the test point based on an objective function related to LLE. All out-of-sample extension methods have a significantly worse inconsistency on the synthetic swiss-roll - where MVU is very consistent. On real world data sets, such as handwritten *fives* from the MNIST data set, facial images from the frey data set and the rotating teapot data set their relative inconsistency is always less than twice that of MVU. In the case of the teapot data set all three out-of-sample extensions have even smaller inconsistency than MVU. It is interesting to notice that the LLE inspired method has lower inconsistency than both sdp-based approaches throughout all data sets.

The conjugate gradient post-processing improves the consistency significantly on the synthetic swiss-roll (which is sampled from a perfect manifold) but does not improve the results on any of the three real-world data sets.

3.1.11 Maximum vs Minimum Trace

One interesting aspect of maximum variance unfolding is that it tries to find a low rank inner-product matrix by maximizing its trace. Without the constraints, maximizing the

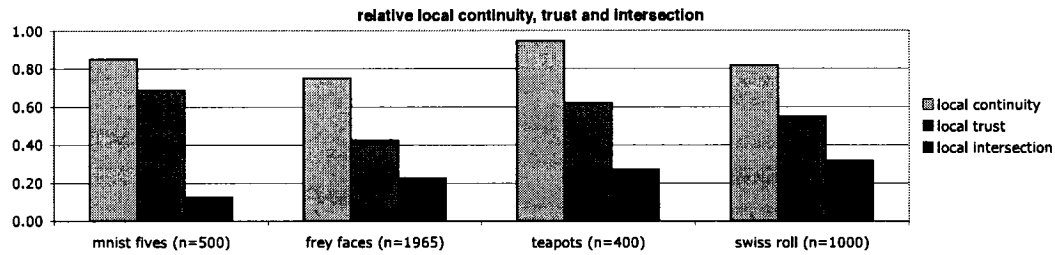


Figure 3.24: Quantitative evaluation of local continuity, trust and intersection. The bar plot shows the results of minimum trace embeddings relative to maximum trace embeddings. On all data sets and under all metrics, mimimum trace MVU is significantly worse than maximum trace MVU.

trace moves the inner-product matrix in the direction of the identity matrix and therefore would *increase* its rank. If the matrix is centered, as it is in the case of MVU, maximizing the trace is identical to maximizing all pairwise distances. Although there exists no rigorous proof that the rank will indeed be low, intuitively maximizing all pairwise distances in presence of local distance constraints will "unfold" the data into a lower dimensional sub-space - as discussed in Section 3.1.4.

Let $\mathbf{K}$ be a symmetric matrix with eigenvalues $\{\lambda_i\}_{i=0}^{n}$. The trace is a common approximation of the rank where the L_0 norm is substituted by the L_1 norm:

$$\text{rank}(\mathbf{K}) = \sum_i \|\lambda_i\|_0, \quad \text{trace}(\mathbf{K}) = \sum_i \|\lambda_i\|_1. \tag{3.35}$$

Clearly, the trace is a much nicer behaved function than the rank. It is linear, continuous and differentiable. It is therefore common practice to minimize the trace of a matrix instead of its rank. In the case of MVU, as we are interested in a low-rank kernel matrix $\mathbf{K}$, would it make sense to minimize the rank instead of maximizing it? This section investigates this question empirically on a selection of various data sets.

First of all, the resulting optimization problem is still a well-defined semi-definite program:

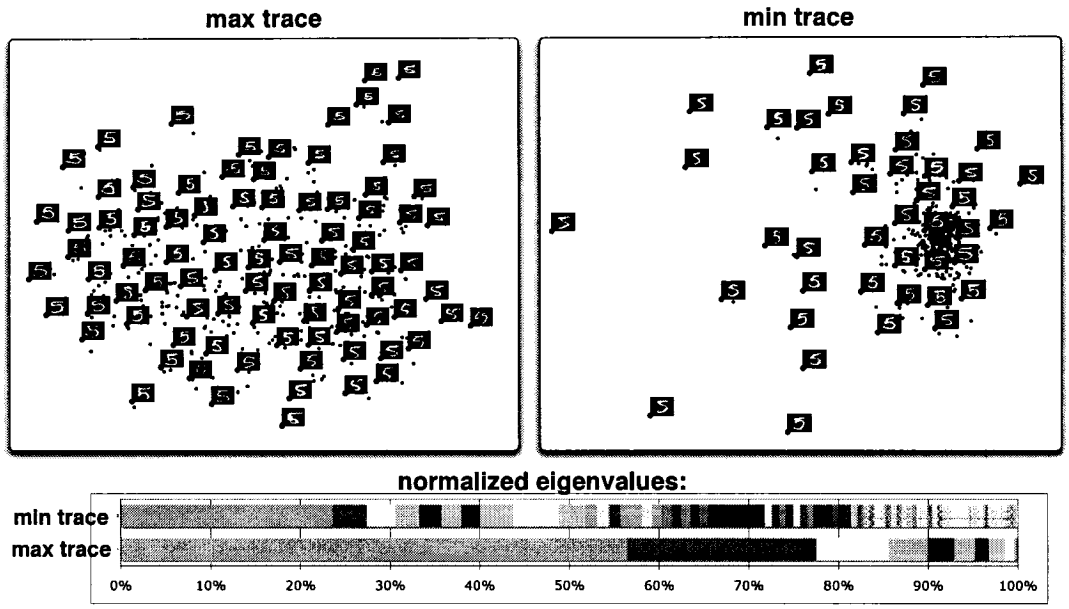


Figure 3.25: Embeddings of MNIST handwritten *fives* with maximum trace (left) and minimum variance unfolding (right). The bottom plot shows the normalized eigenvalues in both cases.

Minimize trace(K) **subject to:**

(1) $\mathrm{K} \succeq 0$.

(2) $\Sigma_{ij}\mathrm{K}_{ij} = 0$.

(3) $\mathrm{K}_{ii} - 2\mathrm{K}_{ij} + \mathrm{K}_{jj} = D_{ij}$ **for all** (i,j) **with** $\eta_{ij} = 1$.

We evaluated the new (minimum trace) SDP on four data sets. To analyze the quality of the embeddings in a quantitative manner, we used the metrics *local continuity*, *local trust*, and *local intersection* from Section 3.1.9. Figure 3.24 shows the evaluation of minimum trace embeddings relative to maximum trace embeddings. A value of 1 would imply that the minimum trace embedding is of equal quality than the maximum trace embedding. The results reveal that all values are less than 1 and therefore minimum variance unfolding is significantly worse than maximum trace MVU under all three metrics and on all the data sets that we tested on.

We used three image data sets of handwritten digits, a rotating object and facial expressions (all previously introduced in Section 3.1.5). Further, we applied the min trace

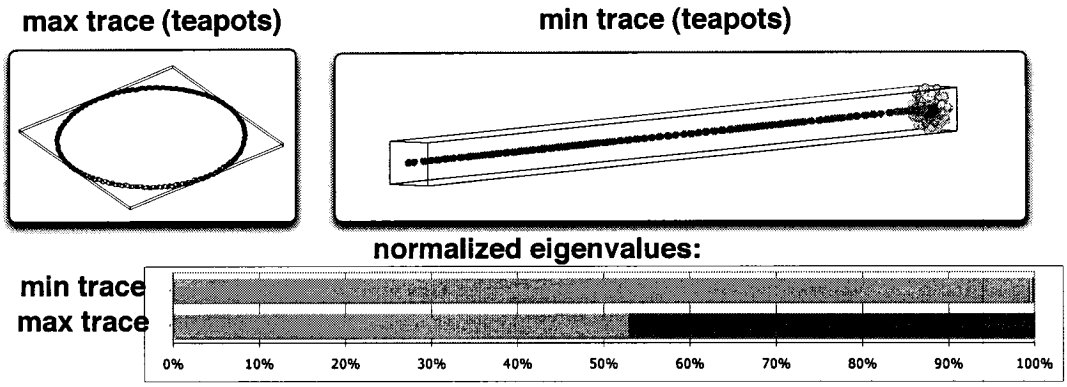


Figure 3.26: The maximum and minimum trace embeddings of the teapot data set. The points are colored according to degree of rotation.

variation of MVU on a noise-free swiss-roll, whose output was almost identical to the input. In the case of the swiss-roll and the teapot data we used only the leading two eigenvectors to evaluate all three metrics in Figure 3.24. On the digits and face data sets we used the leading 10 dimensions. As a general trend, we observed that the minimum trace embeddings have a much higher relative load in their first dimension but tend to have a long tail of small eigenvalues that correspond to many dimensions with low variance. The projection onto a small subspace therefore leads to a much higher error and lower quality embeddings.

Figure 3.25 shows the two embeddings of $n = 500$ handwritten FIVES from the mnist data set. The eigenvalue spectrum clearly shows that minimizing the trace leads to a much higher dimensional output. In fact, there is no clear cutoff point and a projection onto any low dimensional sub-space would lead to very high error. Minimizing the trace is equivalent to minimizing all pairwise distances. The output in the right plot shows that all digits are clamped together in a "ball". This allows neighborhoods to infiltrate each other, even while preserving local distances.

The embeddings of the 360 degree rotation teapot data set are shown in Figure 3.26. In this case, minimum variance unfolding leads to an almost one dimensional embedding and therefore (at first glance) manages to find a lower dimensional representation. However,

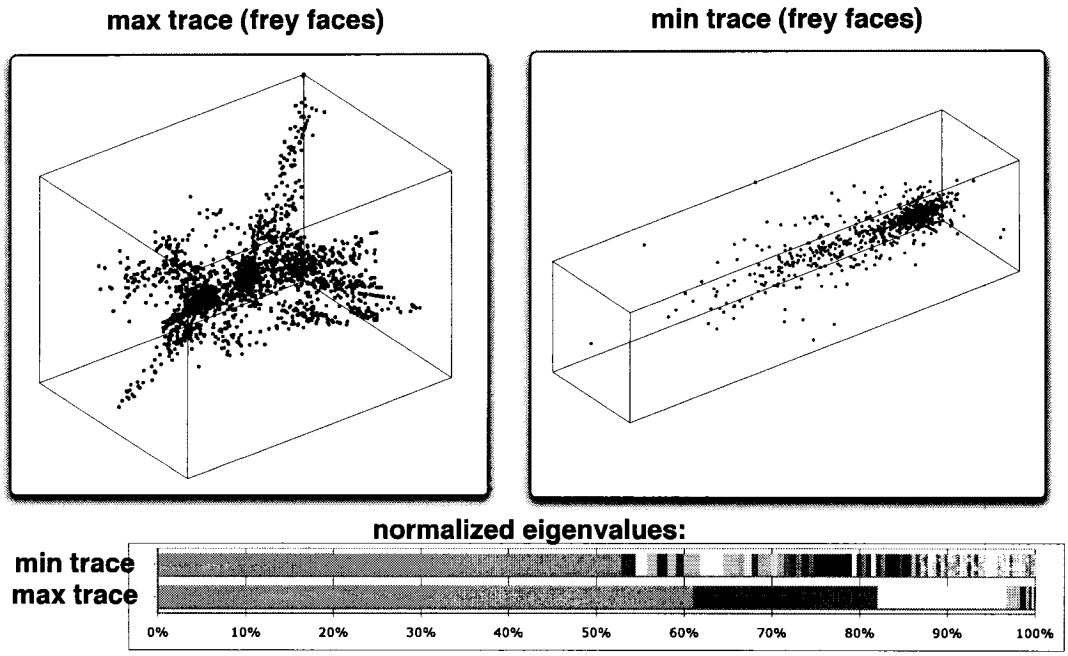


Figure 3.27: Mimimum and maximum trace embeddings of the Frey facial expression data set.

a closer look at Figure 3.26 reveals that significant information is stored in the tailing eigenvectors. A projection of the output onto 1 dimension would collapse the majority of the inputs into a single point. Figure 3.24 also testifies the poor quality of the embedding.

Finally we experimented with the Frey facial expression data set. The results can be seen in Figure 3.24. The right plot shows the minimum trace embedding. In contrast to the left plot the points are more clamped together and the first eigenvalue has more relative loading. This explains the bad quality of the embedding, as shown in Fig. 3.24.

Overall, the results show that minimizing the trace is not as effective as maximizing the trace. Although in some cases the leading eigenvalues tend to bear more load, the overall embedding is of less quality under all of our metrics. One extension of MVU - minimum volume embedding [75] capitalizes on the fact that minimizing the trace tends to result in a few large eigenvalues. The algorithm forces the output to be low dimensional by iteratively projecting onto the leading eigenvectors while minimizing the trace of $\mathbf{K}$.

3.1.12 Practical Aspects of MVU

In practice, the outcome of maximum variance unfolding mostly depends on the underlying neighborhood graph. First, it is important that the neighborhood graph is connected or the SDP is infeasible. Second, the neighborhood graph should not have too many local links, or MVU will "lock up". In that case the high dimensional input is the only data set that satisfies all constraints and the output will be identical to the input. For the experiments in this paper we usually used a $k-$nearest neighbors graph with k varying between $k = 3, 4, 5$. For values of k that are larger than 5, the SDP soon becomes intractable on most data sets of size $n > 1000$. Further, it is an important choice if MVU should be run with our without slack variables.

In our experience, there are two methods to choose k. If the local distances don't have to be preserved exactly, then a large value of k in combination with MVU with slack variables tends to give robust results. Here the trade-off is mostly between embedding quality and CPU time. The slack variables will usually prevent MVU from "locking up".

If one wants to preserve local distances (and angles) exactly, MVU must be used without slack variables. Unfortunately, this means that a value of k that is too large can easily cause "locking up". The best recipe for good results here is to apply MVU several times with increasing k. The lowest value of k should be chosen such that the neighborhood graph is still connected with $k \geq 3$. Values lower than 3 tend to lead to degenerate results (one dimensional strings of points) and should generally be avoided - also, they are rarely connected. To find the right embedding, one can compare the eigenvalue spectra of each run. If k was too large and MVU did lock up, the eigenvalue spectrum generally differs significantly from runs with lower value of k. Figure 3.29 illustrates this on a synthetic Swiss roll example of size $n = 1000$. Here $k = 4, 5, 6$ are clearly the best choices. The bar plot in the bottom left shows the harmonic mean of local trust, local continuity and local intersection of the resulting two dimensional embeddings. The normalized eigenvalue plot in the bottom right reveals that for $k \geq 7$ three dimensions are required with almost equal variance. This is the same eigenvalue spectrum as that of PCA on the input - a clear sign

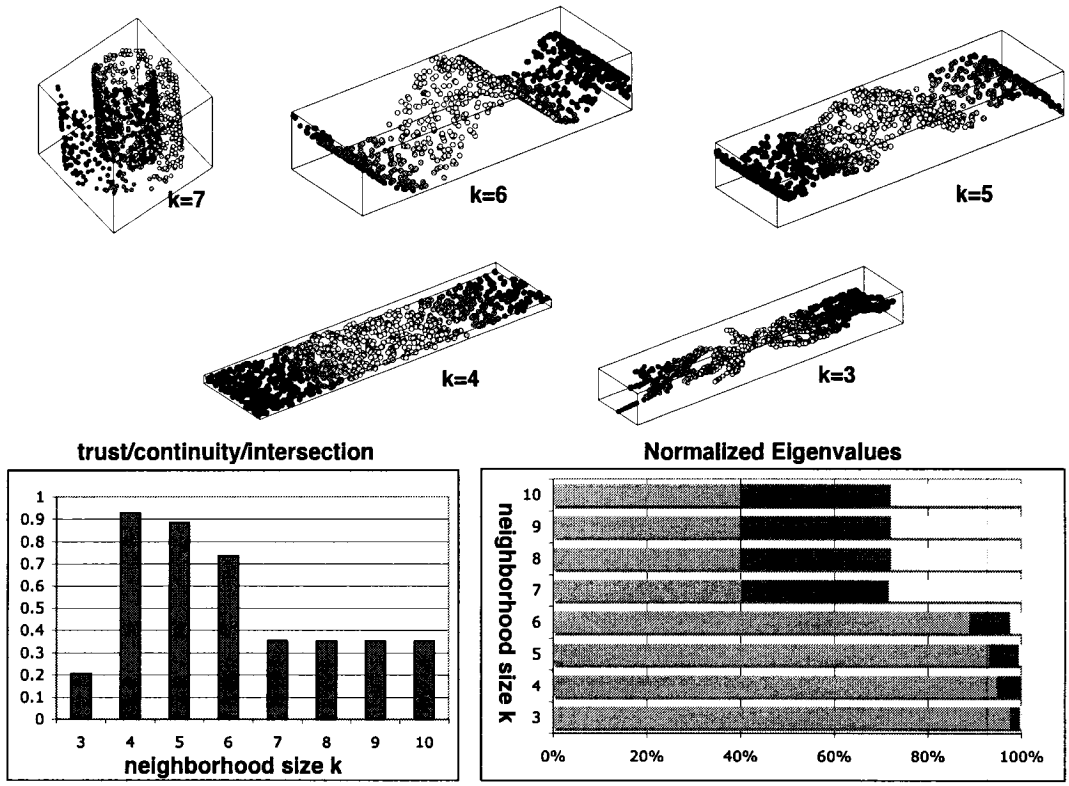


Figure 3.28: A Swiss roll data set ($n = 1000$) unfolded under varying values of k. The data set locks at $k = 7$. The bar plot on the left shows the harmonic mean of local trust, local continuity and local intersection. The normalized eigenvalue spectrum in the bottom right reveals that the data did not unfold with $k \geq 7$.

that the data set did not unfold. The eigenvalues corresponding to $k = 4$ reveal that all variance is kept in the first two dimensions.

If for all values of k the neighborhood graph is disconnected, MVU can be run on all connected components individually. Alternatively, one can also add artificial links between the connected components. Imagine a data set has two connected components C_1 and C_2. We can link them by picking a random point in connected component $x \in C_1$. Let $x' \in C_2$ be the closest point to x in C_2. And let $x'' \in C_1$ be the closest point to x' in C_1. We add an edge between x' and x''. This procedure should be continued until the graph is connected.

MVU is based on a nearest neighbor graph. But what if two distant layers of the

manifold are accidentally connected by a link? See Figure 3.29 for an example of $n = 600$ points sampled from a swiss-roll manifold with additional noise. The left image shows the $k = 7$ nearest neighbor graph with "bad" edges between the outer in the inner layer of the swiss-roll. In this case MVU would preserve the length of those edges and could not unfold the data. One approach to tackle this problem is to detect and cut bad edges. This can be achieved by performing a random walk along the graph with transition probability to get from $\vec{x}_i$ to $\vec{x}_j$ of $p_{ij} = \eta_{ij} \exp(-\frac{\|\vec{x}_i - \vec{x}_j\|}{\sigma})$. Let q_{ij}^t be the probability to get from $\vec{x}_i$ to $\vec{x}_j$ in less than $t + 1$ steps. Bad edges tend to have two identifying features. 1. they are long and 2. there are only few short alternative routes that connect the two nodes. It follows that the probability q_{ij} for a bad edge tends to be much lower than that of a good edge. The right image in Figure 3.29 was generated with $t = 7$ and $\sigma = 1$. The "bad" edges were automatically detected as outliers of the distribution of $q_{ij}^t + q_{ji}^t$ and cut. An alternative method is to use convex flow, as described in [30].

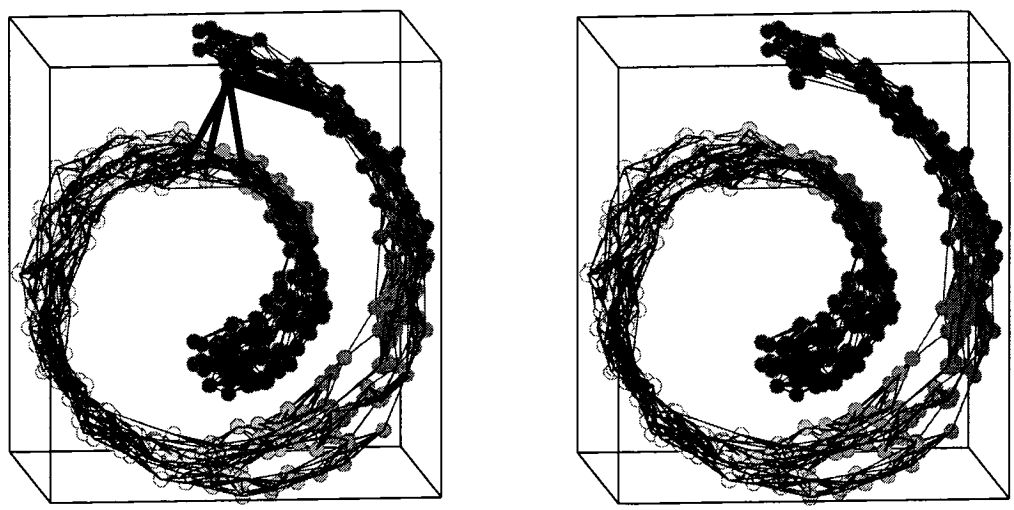


Figure 3.29: A Swiss roll data set with bad edges. The image on the left shows the original $k = 7$ nearest neighbor graph. Edges with low traverse probability are highlighted in black. The right image shows the same graph after the highlighted edges have been cut.

3.2 Related Work

Maximum Variance Unfolding can be viewed as bridging two ongoing lines of research in machine learning, one on spectral methods for dimensionality reduction, the other on kernel methods for pattern recognition. In this section we will briefly review both lines of research and discuss the relationship with MVU accordingly.

3.2.1 Spectral Methods

The last few years have witnessed a number of developments in spectral methods for dimensionality reduction and manifold learning. Recently proposed algorithms include Isomap [82], locally linear embedding (LLE) [66], and Laplacian eigenmaps [5]; there are also related algorithms for clustering [62, 77]. Maximum variance unfolding is based on a different geometric intuition than these other algorithms, however, and as a result, it has somewhat different properties. This section briefly reviews previous algorithms and highlights the similarities and differences.

Most spectral methods for dimensionality reduction all share the same basic structure: (i) computing neighborhoods in the input space, (ii) constructing a square matrix with as many rows as inputs, and (iii) deriving a low dimensional embedding from the top or bottom eigenvectors of this matrix. Algorithms differ in the geometric signatures of manifolds that they attempt to estimate and preserve. For example, Isomap is based on geodesic distances, LLE on the coefficients of local linear reconstructions, and Laplacian eigenmaps on the discrete graph Laplacian. Maximum variance unfolding is based on estimating and preserving local distances and angles. It is most easily compared to Isomap since both algorithms attempt to learn isometric mappings and thus seek the same solution, up to a global rotation. We will now briefly review related spectral methods for manifold learning. All of the algorithms assume that the input data is sampled from a smooth manifold and the first step universally is to construct a nearest neighbor graph of the inputs.

Isomap

Recall from Section 3.1.2 that MDS finds a low-dimensional embedding of the input that preserves the pairwise inner products optimally with respect to the L_2 loss. The inner products are computed exactly from the pairwise distances of the inputs. Preserving inner products is generally a good heuristic for preserving pairwise distances (but leads to a tractable eigenvector problem). Isomap [82] (which stands for Isometric Mapping) can be viewed as a non-linear extension of multi-dimensional scaling (MDS). It is based on the same embedding steps, computes the pairwise inner products between two input vectors from the approximated *geodesic* distance between them. The geodesic distance between two points is the length of the shortest path between them along the manifold. As the manifold is unknown, Isomap approximates the *geodesic* distances as the shortest paths on the neighborhood graph. See Figure 3.30 for an illustration.

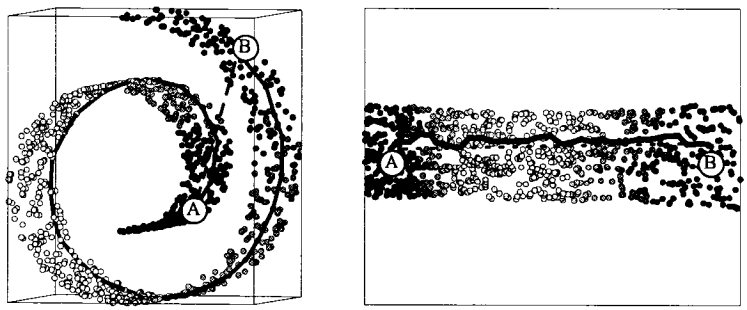


Figure 3.30: *Left:* Isomap approximates the geodesic distance between point A and B (in a Swiss roll data set) as the shortest path between them. *Right:* The output points are aligned such that the Euclidean distance between two points (A and B) resembles the shortest path.

If $\mathbf{D}_{ij} = \|\vec{x}_i - \vec{x}_j\|^2$ is a squared Euclidean distance matrix, the corresponding inner-product matrix can be computed as $\mathbf{G} = -\frac{1}{2}\mathbf{HDH}$, where $\mathbf{H} = \mathbf{I} - \frac{1}{n}\vec{e}\vec{e}^{\top}$ is the centering matrix (with $\vec{e} = [1 \ldots 1]^{\top}$). Let $\vec{y} = [\vec{y}_1, \ldots, \vec{y}_n]$ be the matrix with the output vectors as

columns. Recall that MDS solves the following optimization problem:

$$\textbf{Minimize}_{\vec{y}} \; \|\vec{y}\vec{y}^\top - \mathbf{G}\|_F. \tag{3.36}$$

Let $\hat{\mathbf{D}}_{ij}$ be the squared length of the shortest path on the neighborhood graph between $\vec{x}_i$ and $\vec{x}_j$, and let $\hat{\mathbf{G}} = -\frac{1}{2}\mathbf{H}\hat{\mathbf{D}}\mathbf{H}$ be the corresponding "geodesic inner-product" matrix. Isomap solves the following optimization problem instead:

$$\boxed{\textbf{Minimize}_{\vec{y}} \; \|\vec{y}\vec{y}^\top - \hat{\mathbf{G}}\|_F.}$$

As mentioned in Section 3.1.2, the solution to this optimization problem are the leading eigenvectors of the matrix $-\frac{1}{2}\mathbf{H}\hat{\mathbf{D}}\mathbf{H}$, scaled by their square-rooted eigenvalues. This could be a problem as $\hat{\mathbf{D}}$ is generally not a Euclidean squared distance matrix and some eigenvalues will be negative. In this case, scaling by the square-root of the eigenvalues would lead to complex numbers and a non-Euclidean representation. As a solution, the authors of Isomap suggest to set all negative eigenvalues to zero. It can be shown [15] that this is indeed the optimal solution given the constraint that the output should be Euclidean. The eigenvalues of the inner-product matrix can be used to estimate the underlying dimensionality in the same way as MVU.

To summarize, Isomap proceeds in three steps:

1. Construct a (connected) k-nearest neighbor graph where each edge is labeled with the Euclidean distance between its two nodes.

2. Use Dijkstra's algorithm (or modifications thereof) [1] to compute the squared length of the shortest path $\hat{\mathbf{D}}_{ij}$ between any two input vectors $\vec{x}_i, \vec{x}_j$.

3. Find the eigen-decomposition of $-\frac{1}{2}\mathbf{H}\hat{\mathbf{D}}\mathbf{H} = \mathbf{V}\mathbf{\Delta}\mathbf{V}^\top$ to obtain the Euclidean embedding $\vec{y}_1, \ldots, \vec{y}_n$ where $[\vec{y}_i]_\alpha = \mathbf{V}_{i\alpha}\sqrt{\mathbf{\Delta}_{\alpha\alpha}}$.

On many problems, Isomap and maximum variance unfolding return generally similar results, with eigenvalue spectra that provide a reliable estimate of the data set's intrinsic dimensionality. The key difference between MVU and Isomap is that MVU preserves all

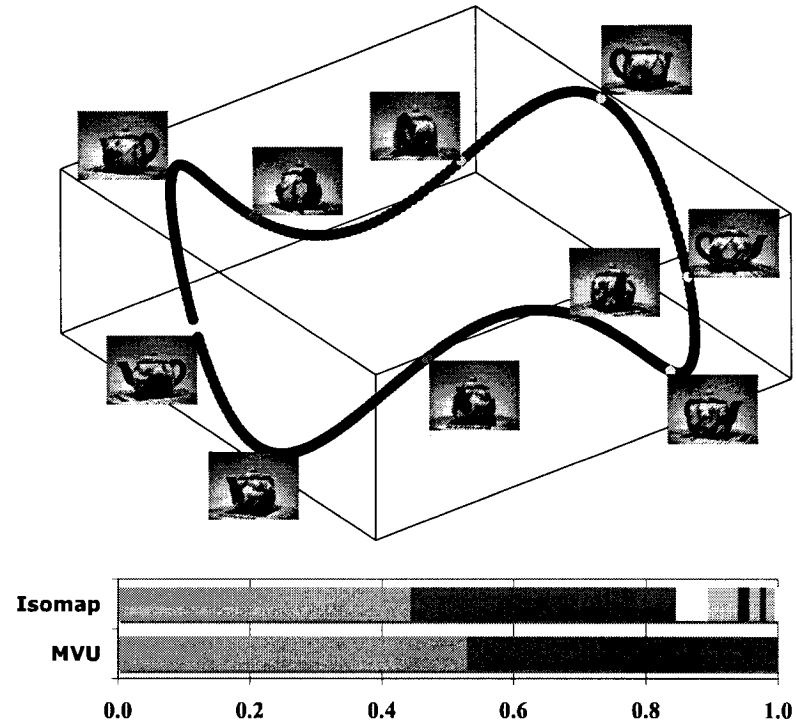


Figure 3.31: The three dimensional embedding of $n=400$ images of a rotating teapot (with accompanying eigenvalue spectrum).

local neighborhood distances whereas Isomap attempts to preserve all *global* pairwise distances. Fig. 3.31 illustrates an example where the algorithms return quite different results. The top panel shows the results of Isomap applied to the same data set of teapot images as in Fig. 3.9; the bottom panel compares its eigenvalue spectrum to that of maximum variance unfolding. Note that Isomap has more than two appreciable eigenvalues (which is manifested by the waves in its three dimensional embedding). This somewhat artificial result is due to the fact that the sampled manifold in this example is not isometric to a *convex* subset of Euclidean space. This is a key assumption of Isomap, one that is not satisfied by many real world data sets [28].

Locally Linear Embedding

Locally Linear Embedding [66] (LLE) is based on a somewhat different intuition than Isomap or Maximum Variance Unfolding. As before, we assume that the input data $\vec{x}_1, \ldots, \vec{x}_n \in \mathcal{R}^d$ is sampled from a manifold and the first step of LLE is to create a neighborhood graph. Instead of preserving local or global distances, LLE focusses on local reconstruction. Let $\mathcal{N}_i$ be the nearest neighbors of $\vec{x}_i$. As manifolds are locally linear, each data point $\vec{x}_i$ can be linearly reconstructed from its nearest neighbors. We can therefore find weights $\mathbf{W}_{ij}$ such that $\vec{x}_i \approx \sum_j \mathbf{W}_{ij}\vec{x}_j$ (where $\mathbf{W}_{ij} = 0$ if $\vec{x}_j \notin \mathcal{N}_i$, ie we only reconstruct locally). The second step of LLE is to find this weight matrix $\mathbf{W}$ that minimizes the loss function

$$E(\mathbf{W}) = \sum_i \|\vec{x}_i - \sum_j \mathbf{W}_{ij}\vec{x}_j\|^2. \tag{3.37}$$

subject to the constraints that for each i, $\sum_j \mathbf{W}_{ij} = 1$ and $\mathbf{W}_{ij} = 0$ if $\vec{x}_j \notin \mathcal{N}_i$. The second constraint ensures that the reconstruction is only local. The first constraint ensures that the reconstruction is invariant to translation and rotation. In other words, if $\bar{x}_i = \mathbf{Q}\vec{x}_i + \vec{t}$, for some rotation matrix $\mathbf{Q}$ and some vector $\vec{t}$, the loss in equation (3.37) with respect to $\bar{x}_i$ would be unchanged.

The third and final step of LLE is to find low dimensional output vectors that allow the same local reconstructions as the input vectors. The goal is to find vectors $\vec{y}_i$ that minimize the loss function

$$E(\vec{y}_1, \ldots, \vec{y}_n) = \sum_i \|\vec{y}_i - \sum_j \mathbf{W}_{ij}\vec{y}_j\|^2, \tag{3.38}$$

subject to the constraints that $\sum_i \vec{y}_i = 0$ and $\frac{1}{n}\sum_i \vec{y}_i\vec{y}_i^\top = \mathbf{I}$. The first constraint centers the output. The second constraint forces the output dimensions to be uncorrelated and to have unit variance. Note that this constraint ensures that not all dimensions can satisfy (3.38) trivially by letting every vector be constant.

Let $Y = [\vec{y}_1, \ldots, \vec{y}_n]$ be the matrix with all output vectors as columns. Using $\vec{y}$, we

can rewrite (3.38) as

$$E(Y) = \|Y^\top - \mathbf{W}Y^\top\|_F^2 = \text{trace}(Y\mathbf{M}Y^\top) \tag{3.39}$$

where $\mathbf{M} = (\mathbf{I} - \mathbf{W})^\top(\mathbf{I} - \mathbf{W})$.[2] The final optimization problem becomes (without the constraint that the output is centered):

$$\boxed{\textbf{Minimize}_Y \text{ trace}(Y\mathbf{M}Y^\top) \textbf{ subject to: } \tfrac{1}{n}YY^\top = \mathbf{I}.}$$

It is easy to show that the solution to this optimization problem consists of the bottom[3] eigenvectors of M [69]. The very bottom eigenvector is the all-constant vector with eigenvalue zero. This follows straight from the fact that each row and column of M is normalized by construction. Also M is positive semidefinite and cannot have negative eigenvalues. (This eigenvector corresponds to the trivial solution of mapping all outputs to the same point and is excluded from the output.) It follows that all remaining eigenvectors are orthogonal to it and are therefore automatically centered.

Like MVU and Isomap, LLE also obtains the final embedding from eigenvectors of a carefully constructed square matrix with as many rows as input vectors. However, in contrast, the matrix M is sparse, which allows the eigenvector computation to be much more efficient. Another significant difference is that the output coordinates are obtained from the *unscaled bottom* eigenvectors and not the re-scaled *leading* eigenvectors - as in MVU or Isomap.

Laplacian Eigenmaps

Laplacian Eigenmaps [4] is a non-linear dimensionality reduction algorithm that is based on the Graph Laplacian. Similar to all previously discussed algorithms, the first step is to create a neighborhood graph. Every edge between two inputs $\vec{x}_i$ and $\vec{x}_j$ has a positive weight $\mathbf{W_{ij}}$ associated with it. The weight encodes the degree of the similarity (a larger weight means the nodes are more similar). Common choices are constant weights (eg

[2]Here we made use of the fact that $\textbf{trace}(\mathbf{A}^\top\mathbf{A}) = \textbf{trace}(\mathbf{A}\mathbf{A}^\top)$ for any real symmetric matrix $\mathbf{A}$.

[3]By bottom eigenvector we mean the one corresponding to the smallest eigenvalue.

$W_{ij} = 1$) or the exponentially decaying heat kernel $\mathbf{W_{ij}} = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{\sigma^2}\right)$. To simplify notation, we will assume that all remaining entries of $\mathbf{W}$ (that do not correspond to edges) are filled with zeros. Let $\mathbf{D}$ be the diagonal matrix defined by

$$\mathbf{D_{ij}} = \begin{cases} \sum_j \mathbf{W}_{ij}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}. \tag{3.40}$$

The (weighted) Graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{W}$. Note that $\mathbf{L}$ is typically a very sparse matrix (most entries do not correspond to edges). The diagonal of $\mathbf{L}$ consists of the weighted degree of each node and all off-diagonal entries are either zero or negative.

The intuition behind Laplacian Eigenmaps is to find a mapping from the graph to the Euclidean space, which maps connected nodes close together. Let $\vec{y}_1, \ldots, y_n$ be the output embedding, we will minimize the following function:

$$\mathcal{L}(\vec{y}_1, \ldots, \vec{y}_n) = \frac{1}{2} \sum_{ij} \mathbf{W}_{ij} \|\vec{y}_i - \vec{y}_j\|^2. \tag{3.41}$$

(The factor $\frac{1}{2}$ is primarily to simplify the following math and does not affect the optimization.) Equation (3.41) penalizes the distances between points that have an edge with a large weight between them. In other words, nodes that are originally close will also be placed close together in the output space. Let $Y = [\vec{y}_1 \ldots \vec{y}_n]$, be the matrix of all output vectors. We can rewrite the loss function as

$$\begin{aligned} \frac{1}{2} \sum_{ij} \mathbf{W}_{ij} \|\vec{y}_i - \vec{y}_j\|^2 = & \sum_i \mathbf{D}_{ii} \vec{y}_i^\top \vec{y}_i - \sum_{ij} \mathbf{W}_{ij} \vec{y}_i^\top \vec{y}_j & (3.42) \\ = & \sum_{ij} \mathbf{L}_{ij} \vec{y}_i^\top \vec{y}_j & (3.43) \\ = & \text{trace}(Y \mathbf{L} Y^\top). & (3.44) \end{aligned}$$

Please note that because equation (3.41) is non-negative for all choices of $\vec{y}_i$, it follows from (3.44) that the matrix $\mathbf{L}$ is positive semidefinite. One obvious shortcoming off the optimization so far is that it allows the trivial solution of $Y = 0$. To avoid this (and to fix the scale) the authors of [4] suggest to add the constraint $Y \mathbf{D} Y^\top = 1$. We thus arrive at the optimization problem:

$$\boxed{\textbf{Minimize}_Y \text{ trace}(YLY^\top) \textbf{ subject to: } YDY^\top = 1}$$

An optimal Y for this optimization problem is given by the solution to the generalized eigenvalue problem:

$$LY^\top = \lambda DY^\top. \tag{3.45}$$

Similar to LLE, the computationally most expensive part of Laplacian Eigenmaps is the eigen-decomposition of a sparse positive semidefinite matrix. This is much faster than the exact versions of Isomap or Maximum Variance Unfolding (for both algorithms there are faster approximate versions [26],[89] that trade off accuracy for speed-up).

Summary on spectral methods

Overall, the different spectral algorithms for manifold learning should be viewed as complementary; each has its own advantages and disadvantages. LLE, and Laplacian eigenmaps construct sparse matrices, and as a result, they are easier to scale to large data sets. On the other hand, their eigenvalue spectra do not reliably reveal the underlying dimensionality of sampled manfolds [69], as do Isomap and maximum variance unfolding. There exist convergence proofs for Isomap [28, 82, 94], but not for the other algorithms. On the other hand, maximum variance unfolding by its very nature provides finite-size guarantees that its constraints will lead to locally isometric embeddings. We are not aware of any finite-size guarantees provided by the other algorithms. Finally, while the different algorithms have different computational bottlenecks, it is fair to say that maximum variance unfolding, based on semidefinite programming, is the most computationally demanding.

3.2.2 Kernel Methods

Along with the growing interest in manifold learning, the last few years have also witnessed an explosion of interest in kernel methods for pattern recognition [70]. Kernel methods rely on an implicit mapping of inputs to a higher (and potentially infinite) dimensional feature space. The kernel function specifies the dot product between the feature

vectors formed in this way from the original inputs. Although most of the recent literature on kernel methods focusses on classification and regression [60], there is also very interesting work on kernel PCA for dimensionality reduction [71]. In the remainder of this chapter, we will show that MVU can be understood as an algorithm to learn a kernel matrix for kernel PCA.

Kernel PCA

Schölkopf, Smola, and Müller [71] introduced kernel PCA as a nonlinear generalization of PCA [44]. The generalization is obtained by mapping the original inputs into a higher (and possibly infinite) dimensional feature space $\mathcal{F}$ before extracting the principal components. In particular, consider inputs $\vec{x}_1, \ldots, \vec{x}_N \in \mathcal{R}^D$ and features $\Phi(\vec{x}_1), \ldots, \Phi(\vec{x}_N) \in \mathcal{F}$ computed by some mapping $\Phi : \mathcal{R}^D \rightarrow \mathcal{F}$. Kernel PCA is based on the insight that the principal components in $\mathcal{F}$ can be computed for mappings $\Phi(\vec{x})$ that are only implicitly defined by specifying the inner product in feature space—that is, the kernel function $K(\vec{x}, \vec{y}) = \Phi(\vec{x}) \cdot \Phi(\vec{y})$.

Kernel PCA can be used to obtain low dimensional representations of high dimensional inputs. For this, it suffices to compute the dominant eigenvectors of the kernel matrix $K_{ij} = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$. The kernel matrix can be expressed in terms of its eigenvalues λ_α and eigenvectors $\vec{v}_\alpha$ as $K = \sum_\alpha \lambda_\alpha \vec{v}_\alpha \vec{v}_\alpha^T$. Assuming the eigenvalues are sorted from largest to smallest, the d-dimensional embedding that best preserves inner products in feature space is obtained by mapping the input $\vec{x}_i \in \mathcal{R}^D$ to the vector $\vec{y}_i = (\sqrt{\lambda_1} v_{1i}, \ldots, \sqrt{\lambda_d} v_{di})$.

The main freedom in kernel PCA lies in choosing the kernel function $K(\vec{x}, \vec{y})$ or otherwise specifying the kernel matrix K_{ij}. Some widely used kernels are the linear, polynomial and Gaussian kernels, given by:

$$K(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}, \tag{3.46}$$

$$K(\vec{x}, \vec{y}) = (1 + \vec{x} \cdot \vec{y})^p, \tag{3.47}$$

$$K(\vec{x}, \vec{y}) = e^{\frac{-|\vec{x}-\vec{y}|^2}{2\sigma^2}}. \tag{3.48}$$

The linear kernel simply identifies the feature space with the input space. Implicitly, the polynomial kernel maps the inputs into a feature space of dimensionality $O(D^p)$, while the Gaussian kernel maps the inputs onto the surface of an infinite-dimensional sphere.

The dominant eigenvalues of the kernel matrix K_{ij} measure the variance along the principal components in feature space, provided that the features are centered on the origin. The features can always be centered by subtracting out their mean—namely, by the transformation $\Phi(\vec{x}_i) \leftarrow \Phi(\vec{x}_i) - \frac{1}{N}\sum_j \Phi(\vec{x}_j)$. When the mapping $\Phi(\vec{x})$ is only implicitly specified by the kernel function, the "centering" transformation can be applied directly to the kernel matrix. In particular, recomputing the inner products $K_{ij} = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ from the centered features gives:

$$K_{ij} \leftarrow K_{ij} - \frac{2}{N}\sum_k K_{kj} + \frac{1}{N^2}\sum_{k\ell} K_{k\ell}. \tag{3.49}$$

For a centered kernel matrix, the relative weight of the leading d eigenvalues, obtained by dividing their sum by the trace, measures the relative variance captured by the leading d eigenvectors. When this ratio is nearly unity, the data can be viewed as inhabiting a d-dimensional subspace of the feature space, or equivalently, a d-dimensional manifold of the input space.

Learning the kernel matrix

The choice of the kernel plays an important role in kernel PCA, in that different kernels are bound to reveal (or conceal) different types of low dimensional structure. The inner product matrix $\mathbf{K}$, computed by MVU, is guaranteed to be positive semidefinite and can naturally be viewed as a kernel matrix between inputs. In other words, MVU learns a kernel matrix that reveals when high dimensional inputs lie on or near a low dimensional manifold. The last step of MVU, to obtain the output through a diagonalization of the inner-product matrix $\mathbf{K}$, is identical to kernel PCA. In the remainder of this section, we will view MVU as an algorithm to provide a kernel $\mathbf{K}$ for kernel PCA.

One similarity to previous work on kernel learning for support vector machines (SVMs)

[36, 50] is that MVU is cast as an instance of semidefinite programming. Semidefinite programming is a natural choice for kernel learning, because a real matrix is a kernel matrix if and only if it is positive semidefinite. The similarity ends there, however, as the optimization criteria for nonlinear dimensionality reduction differ substantially from the criteria for large margin classification.

While there have been attempts to interpret the matrices constructed by Isomap and LLE as kernels [8, 37, 70], their interpretation is less straightforward [91]. The kernel matrix in maximum variance unfolding is interesting in several respects. First, it is based on variance maximization, as opposed to margin maximization [70, 49]; the former applies to unsupervised learning, whereas the latter requires (at least some) labeled examples. Second, whereas most kernel functions are chosen to map the inputs into a higher dimensional feature space, the kernel matrix in maximum variance unfolding does just the opposite, typically mapping the inputs into a lower dimensional space. In the following we will compare MVU kernels with various standard kernels. First we will evaluate their suitability for dimensionality reduction with kernel PCA and then large margin classification with support vector machines.

Dimensionality reduction with kernel PCA

We performed kernel PCA with linear, polynomial, Gaussian, and MVU kernels on data sets where we knew or suspected that the high dimensional inputs were sampled from a low dimensional manifold. Where necessary, kernel matrices were centered before computing principal components, as in eq. (3.49).

In the first experiment, we sampled $n = 800$ inputs from the three dimensional "Swiss roll" data set from Section 3.1.5 (with additive low variance noise). Fig. 3.32 shows the original inputs (top left) and the low dimensional representations discovered by kernel PCA with the use of four different kernels. The MVU kernel was learned with $k = 4$ nearest neighbors. Note that only the kernel matrix learned by MVU has two dominant eigenvalues, indicating the correct underlying dimensionality of the Swiss roll, whereas

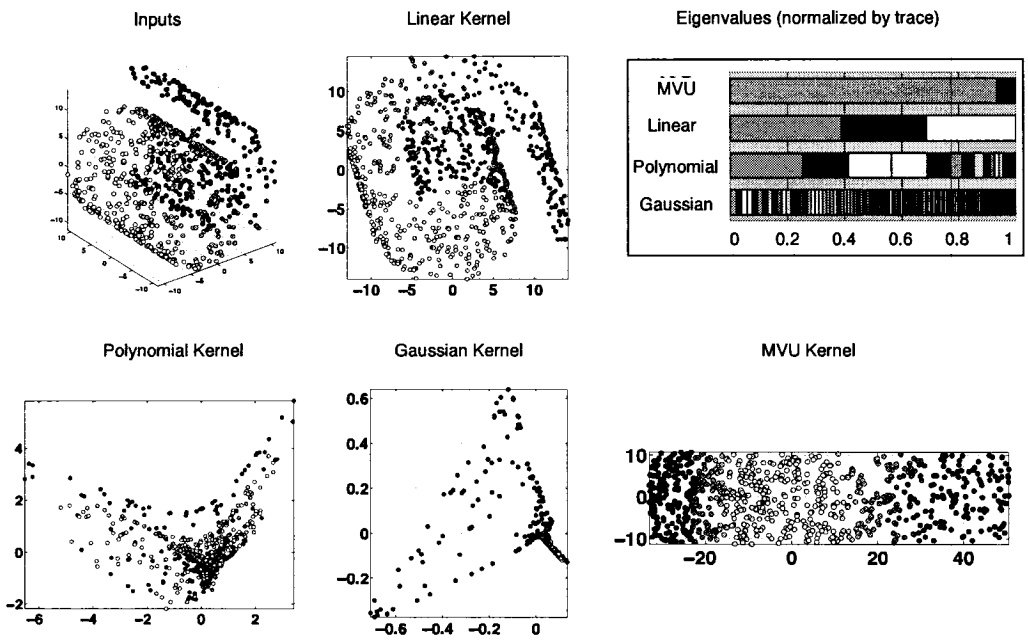


Figure 3.32: The result of kernel PCA on a Swiss Roll data set under various kernels. The MVU kernel is the only one that learns the underlying two dimensional manifold.

the eigenspectra of other kernel matrices fail to reveal this structure. In particular, the linear kernel matrix has three dominant eigenvalues, reflecting the extrinsic dimensionality of the Swiss roll, while the eigenspectra of the polynomial ($p=4$) and Gaussian ($\sigma=1.45$) kernel matrices[4] indicate that the variances of their features $\Phi(\vec{x}_i)$ are spread across a far greater number of dimensions than the original inputs $\vec{x}_i$.

The second experiment was performed on the teapot data set from Section 3.1.5. Figure 3.33 shows the low dimensional embeddings of these images obtained under various kernels. The kernel matrix learned by MVU (with $k=4$ nearest neighbors) concentrates the variance of the feature space in two dimensions and maps the images to a circle, a highly intuitive representation of the full 360 degrees of rotation. By contrast, the linear, polynomial ($p=4$), and Gaussian ($\sigma=1541$) kernel matrices lead to low dimensional representations that do not reveal any underlying structure. Also, their eigenvalue spectra

[4]For all the data sets in this section, we set the width parameter σ of the Gaussian kernel to the estimated standard deviation within neighborhoods of size $k=4$, thus reflecting the same length scale used in MVU.

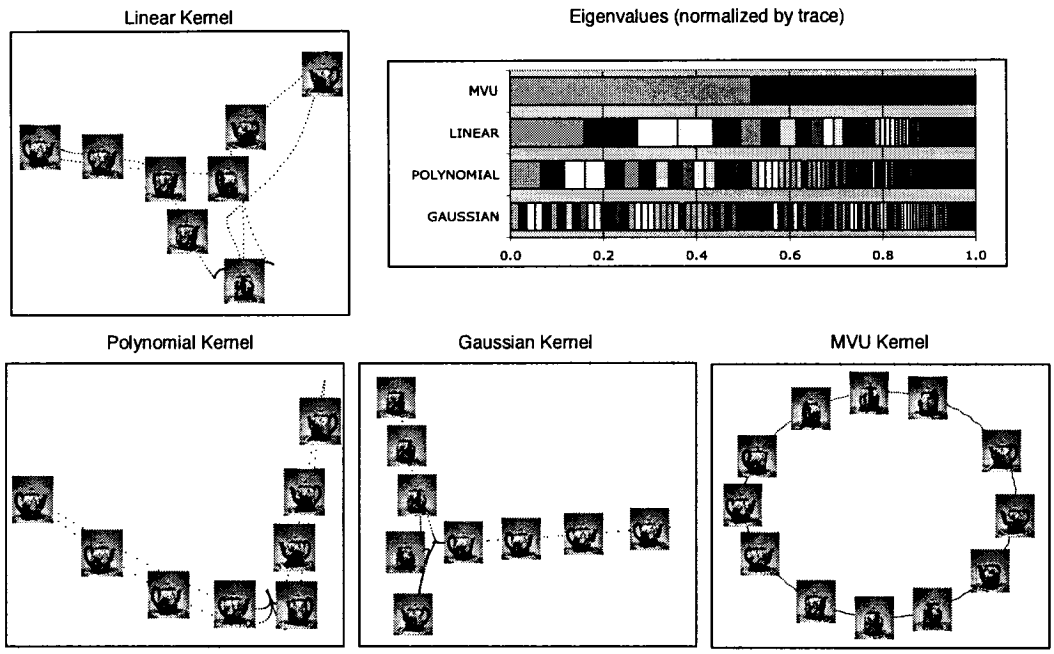


Figure 3.33: The low dimensional representations and eigenvalue spectra for the teapot data from four different kernels. The MVU kernel is the only one that captures an intuitive representation underlying variability.

that do not reflect the low intrinsic dimensionality of the data set.

Why does kernel PCA with the Gaussian kernel perform so differently on these data sets when its width parameter σ reflects the same length scale as neighborhoods in MVU? Note that the Gaussian kernel computes a nearly zero inner product ($K_{ij} \approx 0$) in feature space for inputs $\vec{x}_i$ and $\vec{x}_j$ that do not belong to the same or closely overlapping neighborhoods. It follows from these inner products that the feature vectors $\Phi(\vec{x}_i)$ and $\Phi(\vec{x}_j)$ must be nearly orthogonal. As a result, the different patches of the manifold are mapped into orthogonal regions of the feature space: see Fig. 3.34. Thus, rather than unfolding the manifold, the Gaussian kernel leads to an embedding whose dimensionality is equal to the number of non-overlapping patches of length scale σ. This explains the generally poor performance of the Gaussian kernel for manifold learning (as well as its generally good performance for large margin classification, discussed in the following section).

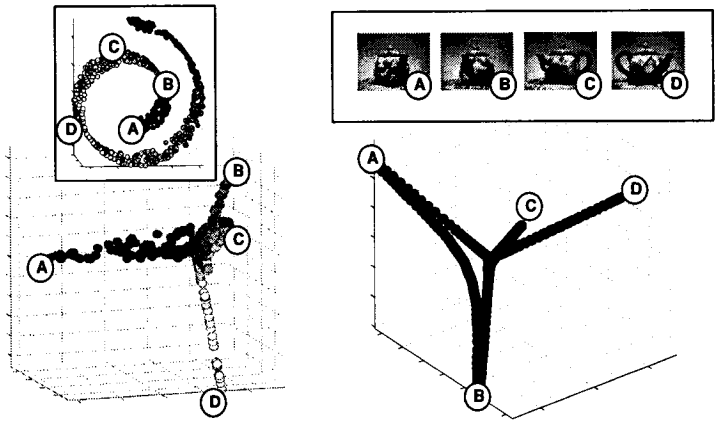


Figure 3.34: Embeddings from kernel PCA with the Gaussian kernel on the Swiss roll and teapot data sets in Figs. 3.32 and 3.33. The first three principal components are shown. In both cases, different patches of the manifolds are mapped to orthogonal parts of feature space.

As a final experiment, we compared the performance of the different kernels on a real-world data set described by an underlying manifold. The data set [40] consisted of $N = 953$ grayscale images at 16×16 resolution of handwritten TWOS and THREES (in roughly equal proportion). In this data set, it is possible to find a relatively smooth "morph" between any pair of images, and a relatively small number of degrees of freedom describe the possible modes of variability (e.g. writing styles for handwritten digits). Fig 3.35 shows the results. Note that the kernel matrix learned by MVU concentrates the variance in a significantly fewer number of dimensions, suggesting it has constructed a more appropriate feature map for nonlinear dimensionality reduction.

MVU for large margin classification

We also evaluated the use of MVU kernel matrices for large margin classification by SVMs. Several training and test sets for problems in binary classification were created from the USPS data set of handwritten digits [40]. Each training and test set had 810 and 90 examples, respectively. For each experiment, the MVU kernel matrices were learned (using $k = 4$ nearest neighbors) on the combined training and test data sets, ignoring

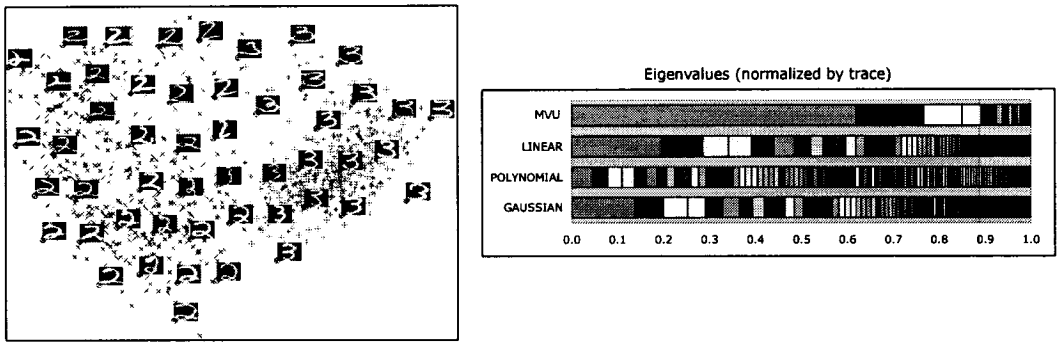


Figure 3.35: Results of kernel PCA applied to $N = 953$ images of handwritten digits. The eigenvalues of different kernel matrices are shown, normalized by their trace.

the target labels. The results were compared to those obtained from linear, polynomial ($p = 2$), and Gaussian ($\sigma = 1$) kernels. Table 3.3 shows that the MVU kernels performed quite poorly in this capacity, even worse than the linear kernels.

Fig. 3.36 offers an explanation of this poor performance. Support vector machines use a linear classifier in feature space. This means that the MVU kernel can only be expected to perform well for large margin classification if the decision boundary on the unfolded manifold is approximately linear. There is no a priori reason, however, to expect this type of linear separability. The example in Fig. 3.36 shows a particular binary labeling of inputs on the Swiss roll for which the decision boundary is much simpler in the input space than on the unfolded manifold. A similar effect seems to be occurring in the large margin

Digits	Linear	Polynomial	Gaussian	MVU
1 vs 2	0.00	0.00	0.14	0.59
1 vs 3	0.23	0.00	0.35	1.73
2 vs 8	2.18	1.12	0.63	3.37
8 vs 9	1.00	0.54	0.29	1.27

Table 3.3: Percent error rates for SVM classification using different kernels on test sets of handwritten digits. Each line represents the average of 10 experiments with different 90/10 splits of training and testing data. Here, the MVU kernel performs much worse than the other kernels.

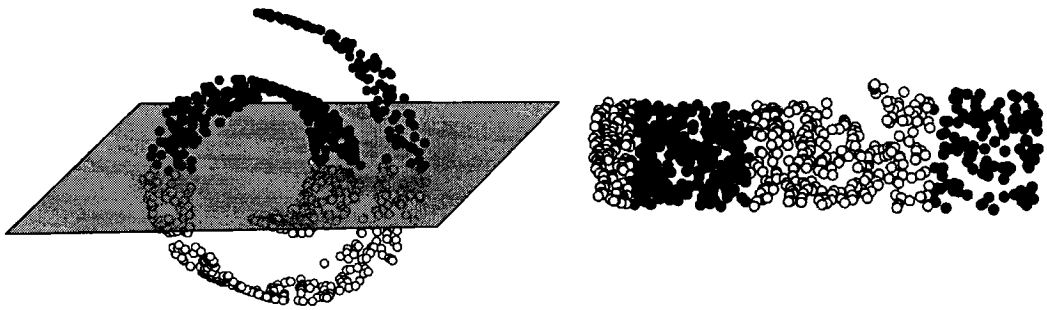


Figure 3.36: *Left:* linearly separable inputs (in black versus white) sampled from a Swiss roll data set. *Right:* unfolding the manifold leads to a more complicated decision boundary.

classification of handwritten digits. Thus, the strength of MVU for nonlinear dimensionality reduction is generally a weakness for large margin classification. By contrast, the polynomial and Gaussian kernels lead to more powerful classifiers precisely because they map inputs to higher dimensional regions of feature space.

Discussion

MVU can be viewed as an unsupervised counterpart to the work of Graepel [36] and Lanckriet et al [50], who proposed learning kernel matrices by semidefinite programming for large margin classification. The kernel matrices learned by MVU differ from those usually employed in SVMs, in that they aim to map inputs into an (effectively) *lower* dimensional feature space. This explains both our positive results in nonlinear dimensionality reduction, as well as our negative results in large margin classification.

MVU can also be viewed as an alternative to manifold learning algorithms such as Isomap [82], locally linear embedding (LLE) [66, 69], and Laplacian eigenmaps [5]. All these algorithms share a similar structure, creating a graph based on nearest neighbors, computing an $N \times N$ matrix from geometric properties of the inputs, and constructing an embedding from the eigenvectors with the largest or smallest nonnegative eigenvalues.

As mentioned earlier, several of the previously mentioned algorithms, such as Isomap

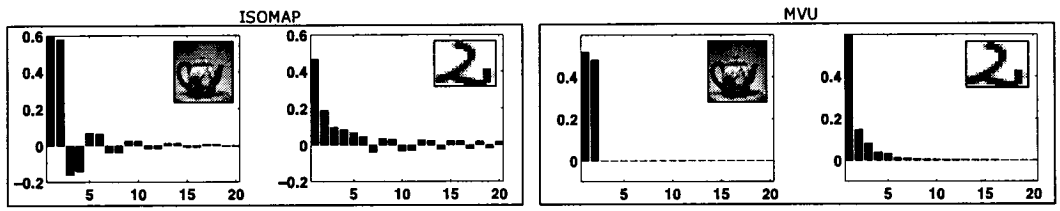


Figure 3.37: The twenty eigenvalues of leading *magnitude* from Isomap and MVU on the data sets of teapot images and handwritten digits. Note that the similarity matrices constructed by Isomap have negative eigenvalues.

and LLE, can also be viewed as kernel methods [38]. However, in general, these other methods give rise to matrices whose geometric properties as kernels are less robust or not as well understood. For example, unlike MVU, the similarity matrix constructed by Isomap from finite data sets can have negative eigenvalues. In some cases, moreover, these negative eigenvalues can be appreciable in magnitude to the dominant positive ones: see Fig. 3.37. Unlike both MVU and Isomap, LLE constructs a matrix whose *bottom* eigenvectors yield low dimensional embeddings; to interpret these matrices as kernels, their eigenvalues must be "flipped", either by inverting the matrix itself or by subtracting it from a large multiple of the identity matrix. Moreover, it does not appear that these eigenvalues can be used to estimate the intrinsic dimensionality of underlying manifolds [69, 72]. A kernel can be derived from the discrete Laplacian by noting its role in the heat diffusion equation, but the intuition gained from this analogy, in terms of diffusion times through a network, does not relate directly to the geometric properties of the kernel matrix. MVU stands apart from these methods in its explicit construction of a semipositive definite kernel matrix that preserves the geometric properties of the inputs up to local isometry and whose eigenvalues indicate the dimensionality of the underlying manifold.

Chapter 4

Conclusion and Future Directions

We presented two algorithms for metric learning for two different settings. For the *supervised* classification scenario, we proposed *large margin nearest neighbor* (LMNN), an algorithm to learn a mapping from input to feature space to improve kNN classification. For the *unsupervised* scenario, we proposed *maximum variance unfolding* (MVU), a non-linear dimensionality reduction algorithm for manifold data sets with fewer intrinsic degrees of freedom than their ambient vector space.

We demonstrated the applicability of both algorithms on many synthetic and real world data sets. The metric learned by the large margin nearest neighbor algorithm makes the nearest neighbor classifier competitive with state of the art machine learning classification algorithms [23]. The extension to multiple metrics improves these results even further, and obtains 1.2% error rate on the MNIST handwritten digits data set. Although even lower error rates have been achieved by designing metrics for $k-$NN classifiers, these other metrics used outside knowledge about the digit data [6, 78]. LMNN in contrast makes no strong assumptions about the type of data and is universally applicable. In contrast to previous methods [34, 35, 76] LMNN focusses only on the local neighborhoods, enforces a large margin and is based on convex optimization. This makes the algorithm very powerful and reliable. Further, we demonstrated that the cost function of LMNN itself can also be used as a classifier. In most cases, this energy-based classifier even outperforms

the $k-$nearest neighbors rule. Another interesting aspect of LMNN is its use for dimensionality reduction. We demonstrated that LMNN can be trained to map input data into a lower dimensional space and thus speed up $k-$NN search by orders of magnitudes. This speedup is further magnified with the use of special, tree-based, data structures [55]. Although the LMNN algorithm is very intuitive and the quantitative results are convincing, a formal proof of convergence is still missing. Future work will focus on a theoretical foundation of the algorithm.

Maximum variance unfolding algorithm finds a low dimensional representation of high dimensional input data. The Euclidean metric in the low dimensional space is often much more meaningful than in the original input space. We demonstrated that the algorithm works successfully on synthetic and real world image manifold data. The heart of MVU involves a semidefinite program, of which we described several variations. These include constraint relaxation via slack variables, inequality constraints, large scale extension via Laplacian graph relaxation and conjugate gradient post-processing. Further, we introduced three objective metrics to quantitatively evaluate and compare MVU with previous work. We refer to the metrics as *local trust*, *local continuity* and *local neighborhood intersection*. On all three metrics MVU is highly competitive with other state-of-the art manifold learning algorithms. In contrast to previous work, maximum variance unfolding has local guarantees and is learned by convex optimization. We further showed that MVU can be viewed as a method to learn a kernel matrix for kernel PCA. An interesting aspect of MVU is that it learns a kernel matrix with maximum trace that is generally low rank. This is in strong contrast to the more common approach [75] to minimize the trace to obtain low rank solutions. A quantitative analysis of both approaches revealed that the maximum trace variant seems to lead to better results under all metrics. Until now there is no rigorous theoretical foundation that proves that trace maximization will lead to low-rank matrices. In fact, we demonstrated on a pathological example that MVU can actually increase the dimensionality of some artificial data sets. As for the large margin nearest neighbor classifier, future work will focus on a stronger theoretical guarantees for

maximum variance unfolding.

Overall, this thesis demonstrates that convex optimization can successfully be used to learn new metrics. We showed that this can be done as a supervised mapping (LMNN) and as an unsupervised embedding (MVU). Both cases involve solving a semidefinite program, which is a special type of convex optimization. We provided detailed instructions of how to implement both algorithms and apply them to large scale data set. MATLAB code for most of the algorithms in this thesis is available at http://www.seas.upenn.edu/~kilianw.

One very interesting question is if a metric can be transfered from one data set to another. One inherent drawback of Maximum Variance Unfolding is that it does not come with an integrated out-of-sample extension. It is an open challenge to learn a precise mapping from the input to the output that generalizes to unknown test sets from the same distribution as the training set.

More challenging is the case where the test set is from a different distribution than the training set. So far we have obtained encouraging preliminary results for the large margin nearest neighbor classifier on two face data sets. Our experiments showed a drastic reduction of the classification error even if the data set used to train the large margin nearest neighbor classifier consisted of different individuals than the set used for kNN classification.

Finally, it would be interesting to explore a unifying framework for both algorithms from this thesis. Besides the obvious similarity that both algorithms learn a metric with convex optimization, MVU and LMNN both preserve or minimize local distances. Could this suggest that there might be a higher level framework which includes both, LMNN and MVU, as a special case?

Bibliography

[1] R. Ahuja, K. Mehlhorn, J.B. Orlin, and R.E. Tarjan. Faster Algorithms for the Shortest Path Problem. *Journal of the Association for Computing Machinery*, 37(2):213–223, 1990.

[2] H. B. Barlow. Unsupervised learning. pages 1–17, 1999.

[3] R.E Barlow, D.J. Bartholomew, J. M. Bremner, and H. D Brunk. *Statistical Inference Under Order Restrictions: The Theory and Application of Isotonic Regression.* Wiley, 1972.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 585–591, Cambridge, MA, 2002. MIT Press.

[5] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (4):509–522, 2002.

[7] J. K. Benedetti. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society*, 39:248–253, 1977.

[8] Y. Bengio, J-F. Paiement, and P. Vincent. Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and spectral clustering. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[9] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.

[10] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1996.

[11] P. Biswas, T.-C. Liang, K.-C. Toh, T.-C. Wang, and Y. Ye. Semidefinite programming approaches for sensor network localization with noisy distance measurements. *IEEE Transactions on Automation Science and Engineering*, 3(4):360–371, 2006.

[12] P. Biswas and Y. Ye. A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization, 2003. Stanford University, Department of Electrical Engineering, working paper.

[13] J. Blitzer, K. Q. Weinberger, L. K. Saul, and F. C. N. Pereira. Hierarchical distributed representations for statistical language modeling. In *Advances in Neural and Information Processing Systems*, volume 17, Cambridge, MA, 2005. MIT Press.

[14] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software 11(1):613-623*, 1999.

[15] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[16] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[17] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[18] L. Chen and A. Buja. *Local Multidimensional Scaling for Nonlinear Dimension Reduction, Graph Layout and Proximity Analysis*. PhD thesis, University of Pennsylvania, July 2006.

[19] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similiarty metric discriminatively, with application to face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-05)*, San Diego, CA, 2005.

[20] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[21] T. Cover and P. Hart. Nearest neighbor pattern classification. In *IEEE Transactions in Information Theory, IT-13*, pages 21–27, 1967.

[22] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[23] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[24] S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss Lemma. *International Computer Science Institute, Technical Report*, pages 99–006, 1999.

[25] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. Technical Report TR-99-006, University of Berkeley, CA, Berkeley, CA, 1999.

[26] V. de Silva and J.B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems*, 15:705–712, 2003.

[27] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. In *Journal of Artificial Intelligence Research*, number 2 in 263-286, 1995.

[28] D. L. Donoho and C. E. Grimes. When does Isomap recover the natural parameterization of families of articulated images? Technical Report 2002-27, Department of Statistics, Stanford University, August 2002.

[29] D. L. Donoho and C. E. Grimes. Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Arts and Sciences*, 100:5591–5596, 2003.

[30] A. Efros, V. Isler, J. Shi, and M. Visontai. Seeing through water. *Advances in Neural Information Processing Systems*, 17:393–400, 2004.

[31] M. Fazel, H. Hindi, and S. P. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *Proceedings of the American Control Conference*, volume 6, pages 4734–4739, June 2001.

[32] Roland A. Fisher. *The Use of Multiple Measures in Taxonomic Problems*. 1936.

[33] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[34] A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems 18*, 2005.

[35] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 513–520, Cambridge, MA, 2005. MIT Press.

[36] T. Graepel. Kernel matrix completion by semidefinite programming. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 694–699. Springer-Verlag, 2002.

[37] J. Ham, D. D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. Technical Report TR-110, Max-Planck-Institut für Biologische Kybernetik, Tübingen, July 2003.

[38] J. Ham, D. D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 369–376, Banff, Canada, 2004.

[39] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18:607–616, 1996.

[40] J. J. Hull. A database for handwritten text recognition research. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994.

[41] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[42] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, 1999.

[43] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. *Contemp. Math*, 26:189–206, 1984.

[44] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[45] LP Kaelbling, ML Littman, and AW Moore. Reinforcement Learning: A Survey. *Arxiv preprint cs.AI/9605103*, 1996.

[46] M.A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[47] J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[48] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 282–289, 2001.

[49] G. R. G. Lanckriet, N. Christianini, P. L. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semi-definite programming. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 323–330, 2002.

[50] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[51] N.D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in Neural Information Processing Systems*, 16:329–336, 2004.

[52] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[53] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. A comparison of learning algorithms for handwritten digit recognition. In F. Fogelman and P. Gallinari, editors, *Proceedings of the 1995 International Conference on Artificial Neural Networks (ICANN-95)*, pages 53–60, Paris, 1995.

[54] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of

learning algorithms for handwritten digit recognition. In F. Fogelman and P. Gallinari, editors, *International Conference on Artificial Neural Networks*, pages 53–60, Paris, 1995. EC2 & Cie.

[55] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 825–832. MIT Press, Cambridge, MA, 2005.

[56] M. Lobo, L. Vandenberghe, S. Boydp, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications: Special Issue on Linear Algebra in Control, Signals and Image Processing*, 284:193–228, 1998.

[57] F. Lu, Y. Lin, and G. Wahba. Robust manifold unfolding with kernel regularization. Technical Report 1108, Department of Statistics, University of Wisconsin-Madison, 2005.

[58] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

[59] T.M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[60] J. Shawe-Taylor N. Christianini. *Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

[61] J. Nash. The imbedding problem for riemannian manifolds. In *Annals of Mathematics*, volume 63, pages 20–63, 1956.

[62] A. Y. Ng, M. Jordan, and Y. Weiss. On spectral clustering: analysis and an algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 849–856, Cambridge, MA, 2002. MIT Press.

[63] S.M. Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1:273–347, 1987.

[64] C. E. Rasmussen and C. K. I Williams. *Gaussian Processes for Machine Learning*. Springer-Verlag, Cambridge, MA, 2006.

[65] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, New Jersey, 1970.

[66] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[67] Agarwal S., Wills J., Cayton L., Lanckriet G., Kriegman D., and Belongie S. Generalized non-metric multidimensional scaling. In Marina Meila and Xiaotong Shen, editors, *Eleventh International Conference on Artificial Intelligence and Statistics*. Omnipress, Puerto Rico, 2007.

[68] R. R. Salakhutdinov and G.E. Hinton. Learning a non-linear embedding by preserving class neighbourhood structure. In Marina Meila and Xiaotong Shen, editors, *Eleventh International Conference on Artificial Intelligence and Statistics*. Omnipress, Puerto Rico, 2007.

[69] L. K. Saul and S. T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

[70] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

[71] B. Schölkopf, A. J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[72] F. Sha and L. K. Saul. Analysis and extension of spectral methods for nonlinear dimensionality reduction. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML 2005)*, Bonn, Germany, 2005.

[73] F. Sha and L. K. Saul. Large margin hidden markov models for automatic speech recognition. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[74] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

[75] B. Shaw and T. Jebara. Minimum volume embedding. In Marina Meila and Xiaotong Shen, editors, *Eleventh International Conference on Artificial Intelligence and Statistics*. Omnipress, Puerto Rico, 2007.

[76] N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment learning and relevant component analysis. In *Proceedings of the Seventh European Conference on Computer Vision (ECCV-02)*, volume 4, pages 776–792, London, UK, 2002. Springer-Verlag.

[77] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 888–905, August 2000.

[78] P. Y . Simard, Y. LeCun, and J. Decker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems*, volume 6, pages 50–58, San Mateo, CA, 1993. Morgan Kaufman.

[79] J. Sun, S. Boyd, L. Xiao, and P. Diaconis. The fastest mixing Markov process on a graph and a connection to a maximum variance unfolding problem, 2004. Submitted to *SIAM Review*.

[80] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[81] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[82] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[83] L. Torresani and K.C. Lee. Large margin component analysis. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[84] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[85] L. Vandenberghe and S. P. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, March 1996.

[86] V. Vapnik. *Statistical Learning Theory*. Wiley, N.Y., 1998.

[87] J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19(6):889–899, 2006.

[88] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

[89] K. Weinberger, F. Sha, Q. Zhu, and L. Saul. Graph laplacian regularization for large-scale semidefinite programming. In B. Schölkopf, J. Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[90] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-04)*, volume 2, pages 988–995, Washington D.C., 2004.

[91] K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 839–846, Banff, Canada, 2004.

[92] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[93] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1537–1544. MIT Press, Cambridge, MA, 2005.

[94] H. Zha and Z. Zhang. Isometric embedding and continuum Isomap. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pages 864–871, 2003.

[95] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimensionality reduction by local tangent space alignment. *SIAM Journal of Scientific Computing*, 26(1):313–338, 2004.