Advancing the Effectiveness of Non-Linear Dimensionality Reduction Techniques

Michael S. Gashler

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

June 2012

ABSTRACT

Advancing the Effectiveness of Non-Linear Dimensionality Reduction Techniques

Michael S. Gashler
Department of Computer Science, BYU
Doctor of Philosophy

Data that is represented with high dimensionality presents a computational complexity challenge for many existing algorithms. Limiting dimensionality by discarding attributes is sometimes a poor solution to this problem because significant high-level concepts may be encoded in the data across many or all of the attributes. Non-linear dimensionality reduction (NLDR) techniques have been successful with many problems at minimizing dimensionality while preserving intrinsic high-level concepts that are encoded with varying combinations of attributes. Unfortunately, many challenges remain with existing NLDR techniques, including excessive computational requirements, an inability to benefit from prior knowledge, and an inability to handle certain difficult conditions that occur in data with many real-world problems. Further, certain practical factors have limited advancement in NLDR, such as a lack of clarity regarding suitable applications for NLDR, and a general inavailability of efficient implementations of complex algorithms.

This dissertation presents a collection of papers that advance the state of NLDR in each of these areas. Contributions of this dissertation include:

- An NLDR algorithm, called Manifold Sculpting, that optimizes its solution using graduated optimization. This approach enables it to obtain better results than methods that only optimize an approximate problem. Additionally, Manifold Sculpting can benefit from prior knowledge about the problem.
- An intelligent neighbor-finding technique called SAFFRON that improves the breadth of problems that existing NLDR techniques can handle.
- A neighborhood refinement technique called CycleCut that further increases the robustness of existing NLDR techniques, and that can work in conjunction with SAFFRON to solve difficult problems.
- Demonstrations of specific applications for NLDR techniques, including the estimation of state within dynamical systems, training of recurrent neural networks, and imputing missing values in data.
- An open source toolkit containing each of the techniques described in this dissertation, as well as several existing NLDR algorithms, and other useful machine learning methods.

Keywords: non-linear dimensionality reduction, manifold learning, intrinsic variables, state estimation, imputation, neighbor selection, neighborhood refinement

ACKNOWLEDGMENTS

I am sincerely grateful to the many people who have played significant roles in helping this work to proceed.

In particular, I thank my advisor, Dr. Tony Martinez, who patiently guided me down the long path of becoming a scholar. I also thank Dr. Dan Ventura and Dr. Christophe Giraud-Carrier, who played significant roles in my education, and in developing parts of this dissertation.

I thank Brigham Young University, and the Computer Science Department in particular, for providing me with a quality education and an unparalleled environment for simultaneously developing my knowledge and character.

I thank Mike Smith and Richard Morris for their contributions to one of the chapters in this dissertation, and to other members of the Neural Networks and Machine Learning Laboratory who have offered advice and insights related to my work.

I am especially grateful to my patient wife, who endured many years of the relative poverty associated with my attending graduate school, who supported me in all of my challenges, who expressed belief in me when I needed it, and without whom none of this would have been possible.

I thank the many scholars upon whose work I have built, and who diligently shared their knowledge so that I could have the opportunity to carry the baton just a little further.

Finally, I acknowledge that the real credit for any good that ultimately comes of this work belongs to my Heavenly Father, who has given me all that I have, who has lovingly supported me in all of my endeavors, and who is the source of all knowledge and wisdom.

Contents

Part I

Overview, Introduction, and Background

Part I provides an overview of this dissertation, and introduces its topic. It is divided into two chapters.

Chapter 1 describes the motivation for this work. It explains the function performed by dimensionality reduction algorithms, and discusses how this technique may be used as an important component in solving many problems.

Chapter 2 gives an overview of related works. It presents a high-level description of other non-linear algorithms for reducing dimensionality, and also of other approaches for using dimensionality reduction to solve problems.

Part II presents improved techniques for performing non-linear dimensionality reduction, or manifold learning, and for making these techniques effective across a diversity of circumstances.

Part III presents works that demonstrate specific applications for manifold learning, and also an open source toolkit containint implementations of various NLDR and manifold learning algorithms.

Part IV concludes by summarizing the contributions of this dissertation, identifying significant problems that remain to be solved in this area, and speculating about further research that is likely to lead to important advances.

Chapter 1

Motivation, and Concept Definitions

Reducing dimensionality is an important function of the human mind. We can reduce visual information encoded by more than 90 million photo-sensitive rods in our retinas to just a few words that succinctly describe what we are looking at. Audio, olfactory, and tactile information are likewise presented to the brain by means of a very large number of nerves, yet we can summarize all of this information with a few adjectives that show we understand what these complex and high-dimensional observations represent. The effectiveness of human memory further implies that reducing information is an important function of human thinking. It naturally follows that reducing the dimensionality of data might be an important operation for computational processes that work with large amounts of data.

As digital information becomes available in ever-increasing abundance, the importance of automated compuational methods for processing this information also grows. Humans increasingly rely on summaries and visualizations of data to make decisions, as it is often no longer practical to manually consume all of the available information. Even many otherwise efficient algorithms become computationally intractable when operating on high-dimensional data. This phenomenon is so common that it has been given the label "the curse of dimensionality" [Bellman, 1961].

In this dissertation, we explore methods that improve the effectiveness of automated methods for reducing dimensionality. These approaches mitigate the curse of dimensionality by summarizing high-dimensional vectors with a low-dimensional representation.

1.1 Dimensionality Reduction

Algorithms that produce a low-dimensional representation of high-dimensional data are called dimensionality reduction algorithms. Formally, let $\mathbf{X}$ be a set of d-dimensional vectors, $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n\}$. A dimensionality reduction algorithm would accept $\mathbf{X}$ as input, and would return a corresponding set, $\mathbf{V}$, of t-dimensional vectors, $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, ..., \mathbf{v}_n\}$, where $t < d$. Each $\mathbf{v}_i \in \mathbf{V}$ can be thought of as a low-dimensional representation, projection, or summary of $\mathbf{x}_i \in \mathbf{X}$. (Since categorical values can be trivially encoded as a vector of real values, and other types of data can often be represented with real values as well, we generally assume that algorithms which operate on vectors of real values are sufficient to handle arbitrary types of data. We will consider a study of the most effective way to encode various types of data using real values to be outside the scope of this work.)

The high-dimensional vectors in $\mathbf{X}$ are typically referred to as "samples" or "observations". The features in $\mathbf{X}$ may be called "input attributes", or "extrinsic variables". The corresponding low-dimensional vectors in $\mathbf{V}$ are referred to as "intrinsic values", "target vectors", or "reduced-dimensional vectors". The attributes in $\mathbf{V}$ may be called "intrinsic variables".

Of particular interest in this dissertation are non-linear dimensionality reduction (NLDR) algorithms. These are algorithms that reduce the dimensionality of data with a non-linear transformation. It is common in many domains for high-dimensional observations to exhibit a high degree of linearity in local regions, but not necessarily across the entire dataset (globally). For example, the earth may be approximately flat in local regions, even though it is globally spherical. As another example, if a digital camera is moved by a very small amount, the resulting image is likely to differ by only a small amount, since most of the image content still depicts the same scene. Thus, the images collected by such a camera generally have a locally-linear relationship to the camera's position and orientation. If, however, the camera is moved by a large amount, there is no reason to suppose that the new scene will have anything in common with the original image. Thus, such images generally do not exhibit global linearity. NLDR algorithms, therefore, typically seek to find a low-dimensional representation of data that exhibits linearity in local regions, but without applying any such constraint globally.

High-dimensional observations that exhibit local linearity can be viewed as sampling the surface of a manifold, a structure of lower dimensionality embedded within higher-dimensional space. An algorithm that seeks to model a manifold from sample observations is called a “manifold learning algorithm”, or “manifold learner”. Technically, manifold learning is more specific than NLDR, because manifold learning implies that a model is trained to represent the underlying manifold represented by the data samples. Such a model could be useful, for example, for mapping out-of-band samples from high-to-low or low-to-high dimensional space. The reader should be aware, however, that the literature on this topic is not always careful to make this distinction. Often, the term manifold learning is used interchangibly with the term NLDR to refer to any technique that reduces dimensionality. Fortunately, an arbitrary NLDR algorithm can be easily converted into a manifold learner by combining it with a regression model. This can be done by first passing $\mathbf{X}$ to the NLDR algorithm to compute $\mathbf{V}$, and then training the regression model to map from $\mathbf{X}$ to $\mathbf{V}$, or from $\mathbf{V}$ to $\mathbf{X}$.

1.2 Applications for NLDR

Around the turn of the century (2000), several NLDR algorithms (which we review in greater detail in the next chapter) were published in conjunction with results that demonstrated the ability to recover high-level concepts, such as the orientation of a head, or the curliness of a hand-written digit, from collections of digital images. These results were significant because no single pixel in any of the images was sufficient by itself to characterize the high-level concept represented in the image, yet the NLDR algorithms were able to accurately summarize these concepts with low-dimensional values. This demonstrated that machines were capable of performing a reduction task that previously only humans were able to perform, using an understanding of the high-level concepts exhibited within the images. It also triggered a flurry of interest in using NLDR as a component in machine learning and data mining tasks.

Dimensionality reduction has been used for a long time to make computationally expensive algorithms more tractable. Recent advances in NLDR have made this technique more effective with problems that involve data on non-linear manifolds. It has also opened the door for applications that require an understanding of high-level concepts within data. For example, NLDR is now being applied for many tasks involving computer vision, character recognition, etc. In Chapter 7, we demonstrate that NLDR can be used effectively for estimating state in dynamical systems. In Chapter 8, we apply NLDR with the application of imputing missing values in data.

1.3 Thesis Statement

Improvements to non-linear dimensionality reduction algorithms can enable them to operate more efficiently, handle ill-behaved manifolds, and remove non-linearities in their estimates of intrinsic variables. These improvements increase their effectiveness with existing dimensionality reduction tasks, and also make them suitable for use with a greater diversity of problems than current approaches can handle, including imputing missing values in data, and modeling the state of dynamical systems.

1.4 Publications

Chapters 3 through 9 of this dissertation consist of works that have been published as a result of this dissertation, and also one that has been submitted for review. The references for these works are listed here, and are numbered according to the chapter in which they appear in this dissertation:

3. Gashler, Michael S. and Ventura, Dan and Martinez, Tony, *Manifold Learning by Graduated Optimization*, IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics, 41, 6, 1458–1470, 2011.

4. Gashler, Michael S. and Giraud-Carrier, Christophe and Martinez, Tony, *Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous*, Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08., 900–905, 2008, Dec., 10.1109/ICMLA.2008.154.

5. Gashler, Michael S. and Martinez, Tony, *Robust Manifold Learning With CycleCut*, Connection Science. 24, 1, pp. 57–69. 2012. DOI: 10.1080/09540091.2012.664122.

6. Gashler, Michael S. and Martinez, Tony, *Tangent Space Guided Intelligent Neighbor Finding*, Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'11, 2617–2624, 2011, IEEE Press, San Jose, California, U.S.A.

7. Gashler, Michael S. and Martinez, Tony, *Temporal Nonlinear Dimensionality Reduction*, Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'11, 1959–1966, 2011, IEEE Press, San Jose, California, U.S.A.

8. Gashler, Michael S. and Smith, Michael S. and Morris, Richard. and Martinez, Tony. *Missing Value Imputation With Unsupervised Backpropagation.* In submission.

9. Gashler, Michael S., *Waffles: A Machine Learning Toolkit*, Journal of Machine Learning Research, MLOSS 12, 2383–2387, 2011, July, pp. 1532–4435, JMLR.org and Microtome Publishing, http://www.jmlr.org/papers/volume12/gashler11a/gashler11a.pdf.

It should also be noted that the publication in chapter 3 builds upon an earlier publication which was included in the author's Masters' thesis, and was not included in this Doctors' dissertation. The reference for that publication is:

Gashler, Michael S. and Ventura, Dan and Martinez, Tony, Iterative Non-linear Dimensionality Reduction with Manifold Sculpting, Advances in Neural Information Processing Systems 20, 513–520, 2008, Platt, J.C. and Koller, D. and Singer, Y. and Roweis, S., MIT Press, Cambridge, MA.

Chapter 2

Related Works

This dissertation builds upon a foundation of much previous research. This work would not have been possible without the diligent efforts of those who discovered and shared knowledge in a variety of fields. This chapter briefly surveys work previously done in areas related to the topics covered in this dissertation. A more targeted and comprehensive review of related works is given in each of the individual chapters that comprise this dissertation.

2.1 Dimensionality Reduction Algorithms

Perhaps the simplest form of dimensionality reduction is feature selection, which involves picking a useful subset of the attributes represented in a dataset. It may not be possible to identify the first use of feature selection because people have been careful to choose relevant attributes for as long as they have worked with structured data, but much work has been done in recent times to improve automated feature selection methods [Dash and Liu, 1997]. One advantage of feature selection methods is that the selected attributes retain their original meaning. In this dissertation, however, we focus on dimensionality reduction methods that combine attributes in a manner such that the intrinsic variables do not correspond directly with any of the extrinsic variables. These methods generally have the advantage of being able to compress more information into fewer dimensions. Additionally, they often form intrinsic variables that encode some high-level concept that was previously only represented using many or all of the extrinsic variables. This phenomenon is a particular focus of this dissertation. The first widely-used dimensionality reduction technique that combined features in this manner is Principal Component Analysis (PCA) [Hotelling, 1933]. PCA finds the linear combinations of attributes that account for the variance within the data in a greedy order of components.

One of the first dimensionality reduction algorithms that utilized a non-linear transformation is Sammon's mapping [Sammon, 1969]. Subsequently, several techniques based on Self-organizing Map (SOM) [Kohonen, 1997] were presented, including Curvilinear Component Analysis and Curvilinear Distance Analysis (CDA)[Demartines and Hérault, 1997, Lee et al., 2000]. CDA is noteworthy in that it utilized a method for estimating geodesic distances by hopping from neighbor-to-neighbor within the data. In 1999, a popular approach called Kernel-PCA [Schölkopf et al., 1999] was presented, which applied the "kernel trick" [Aizerman et al., 1964] within Principal Component Analysis to handle non-linearities. Instead of finding principal components based on the covariance of the data, it finds principal components based on the covariance of augmented data, and it computes this efficiently by means of the kernel trick.

The study of non-linear dimensionality reduction techniques increased dramatically in machine learning communities in 2000 when Isomap [Tenenbaum et al., 2000] and LLE [Roweis and Saul, 2000] were introduced. Isomap utilizes the same technique for estimating geodesic distances as CDA, but computes results using classical multi-dimensional scaling, instead of the iterative approach based on SOM that CDA uses. LLE is similar to Isomap in that it computes results in a single pass, but it differs in the relationships between points that it seeks to preserve. LLE computes each point as a linear combination of its neighboring points. It also has the advantage of being highly optimizable using sparse matrix techniques. In addition to being relatively efficient, Isomap and LLE also presented results that demonstrated the ability to recover high-level (encoded across many attributes) concepts from large datasets. Since that time, many new NLDR algorithms have been presented. Some examples include Local Tangent Space Alignment [Zhang and Zha, 2002], which estimates the tangent-space in local neighborhoods and then seeks a transformation that aligns these tangent spaces, Maximum Variance Unfolding [Weinberger et al., 2004], which utilizes semi-definite programming to maximize the variance within data constrained to preserve distances and angles in local neighborhoods, and Non-linear PCA [Scholz et al., 2005], which uses a variation on backpropagation to train a multi-layer perceptron to fit to a manifold structure. Many other algorithms were also introducted [Belkin and Niyogi, 2001, Donoho and Grimes, 2003, Lafon, 2004, Hinton and Salakhutdinov, 2006, Venna and Kaski, 2006, Zhang and Wang, 2007], which we will not attempt to describe in this dissertation.

2.2 Neighborhood Selection and Refinement

Many NLDR algorithms rely on finding local neighborhoods within sample points to represent the structure of the manifolds they sample. This dissertation also presents techniques for intelligently selecting and refining local neighborhoods for manifold learning. It has been shown that as dimensionality becomes large, the distance from a point to its farthest neighbor approaches the distance to its nearest neighbor [Jonathan et al., 1999]. Thus, intelligent neighbor-selecting techniques become important for manifold learning in high-dimensional space. Numerous alternatative distance metrics have been proposed, but more recent techniques that adaptively analyze the data to select neighbors have shown increased effectiveness with high-dimensional data [Zhou et al., 2004, Wei et al., 2008]. Even when intelligent neighbor-selecting techniques are used, rogue neighbor connections that shortcut across manifold boundaries can cause problems. Techniques have been presented to detect and remove such connections [Cukierski and Foran, 2008].

2.3 Applications

One of the applications for which this dissertation presents a customized dimensionality reduction algorithm is that of modeling a dynamical system. Many papers have proposed techniques that utilize dimensionality reduction to solve the more specific problem of robot tracking [Black, 1996, Crowley et al., 1998, Pourraz and Crowley, 1998]. As advances in manifold learning have been made, several researchers have pointed out that robot observations lie on the surface of a manifold, and that manifold learning can be used to organize this information [Nayar et al., 1996, Keeratipranon et al., 2006].

Another application we visit is that of imputing missing values in datasets. This topic has been well-studied [Jones, 1996, Quinlan, 1989, Lakshminarayan et al., 1996, Farhangfar et al., 2008, Shafer, 1997, Schafer and Graham, 2002, Li et al., 2004, Acuña and Rodriguez, 2004]. The most relevant approaches to this dissertation, however, are those that utilize a form of dimensionality reduction for imputation [Koren et al., 2009, Scholz et al., 2005].

This dissertation also presents an open source toolkit of machine learning algorithms, with particular emphasis on NLDR techniques. The need for such toolkits has been formally identified, and a call made for their development [Sonnenburg et al., 2007]. The toolkit presented in this dissertation was originally released to the public prior to that call, but development has continued since that time to meet the needs of the machine learning community. Several other open source machine learning toolkits have also been developed [Witten and Frank, 2005, Sonnenburg et al., 2010, Zito et al., 2009, King, 2009, Webers et al., 2009, Albanese et al., 2012], but none of them include all of the NLDR techniques that our toolkit covers.

Part II

Improving Non-linear Dimensionality Reduction

The chapters in this part explore methods for improving upon or augmenting the capabilities of existing learning techniques.

Chapter 3 presents a method for NLDR called Manifold Sculpting based on graduated optimization. Most existing NLDR techniques utilize some form of optimization technique to find points in low-dimensional space that exhibit distances, or other relationships, in local neighborhoods similar to those in high-dimensional space. Because of the difficulty of this optimization step, most algorithms use an efficient optimization technique that only finds a solution to an approximate problem. The approach demonstrated in this chapter, however, shows that it is possible to efficiently optimize directly with respect to local distances and other relationships without resorting to an approximate solution. Additionally, this chapter shows that partial supervision can be used to improve the efficiency of this optimization step.

Chapter 4 presents an ensemble technique for improving supervised learning with decision trees. Although dimensionality reduction is generally considered to be an unsupervised process, feature selection is an example of dimensionality reduction that is commonly assisted by class labels. Another example is Chapter 3, which demonstrates that adding supervision to dimensionality reduction can improve performance. Supervised learning methods, therefore, may be viewed as a form of dimensionality reduction that reduces a vector of features to a representative class label. The approach presented in this chaper is significant because it demonstrates classification accuracy that outperforms that of Random Forest, an ensemble approach known for its high accuracy with many datasets [Caruana et al., 2008].

Chapter 5 presents a technique called CycleCut for pruning connections in a graph that interfere with NLDR. Because most NLDR techniques, including Manifold Sculpting (presented in Chapter 3), rely on a graph of neighborhood connections to represent the structure of the manifold sampled by the data points, refining those neighborhood connections can have a significant impact on the results from NLDR. In particular, the approach presented in this chapter enable NLDR to achieve good results with problems that could not be effectively analyzed using previously existing techniques.

Chapter 6 presents a method called SAFFRON for intelligent neighbor-finding. Like CycleCut, SAFFRON also focusses on improving NLDR by improving the graph of neighborhood connections that represent the manifold. SAFFRON, however, focusses on carefully selecting neighbors with desirable properties, rather than pruning them after local neighborhoods have been established. Significantly, SAFFRON can be combined with CycleCut to solve difficult manifolds, including some self-intersecting manifolds. Both approaches can also be used in conjunction with Manifold Sculpting to form a robust NLDR algorithm.

Chapter 3

Manifold Learning by Graduated Optimization

Abstract: We present an algorithm for manifold learning called *Manifold Sculpting*, which utilizes graduated optimization to seek an accurate manifold embedding. Empirical analysis across a wide range of manifold problems indicates that Manifold Sculpting yields more accurate results than a number of existing algorithms, including Isomap, LLE, HLLE, and L-MVU, and is significantly more efficient than HLLE and L-MVU. Manifold Sculpting also has the ability to benefit from prior knowledge about expected results.

3.1 Introduction

Large dimensionality is a significant problem for many machine learning algorithms [Bellman, 1961]. Dimensionality reduction algorithms address this issue by projecting data into fewer dimensions while attempting to preserve as much of the informational content in the data as possible.

Dimensionality reduction involves transforming data to occupy as few dimensions as possible so that the other dimensions may be eliminated with minimal loss of information. Nonlinear transformations have more flexibility to align the data with a few dimensional axes, but also have more potential to disrupt the structure of the data in that process. Manifold learning algorithms seek a balance by prioritizing the preservation of data structure in local neighborhoods. A projection is deemed to be good if the relationships (typically

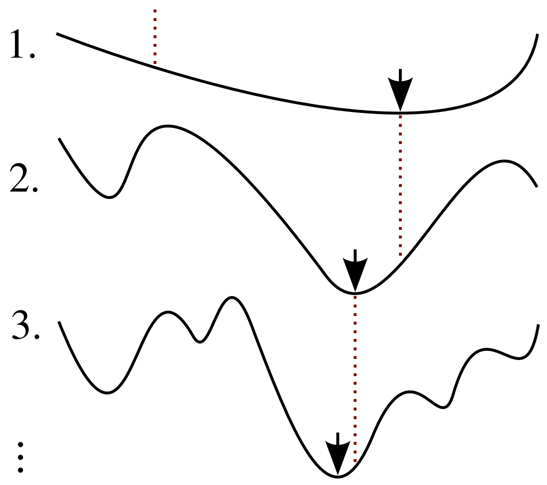


Figure 3.1: With graduated optimization, the solution to each optimization problem gives a good starting point for the next harder problem. If the optimum of the first problem is found, and the solution to each problem is within the convex region around the optimum of the next problem, then graduated optimization will find the optimum of the final problem.

distances and/or angles) between neighboring points after the projection are very similar to the relationships between those same points before the projection.

Manifold learning, therefore, requires solving an optimization problem. In general, global optimization over a non-linear error surface is an NP-hard problem [Cheng and Kovalyov, 2002]. Most popular manifold learning algorithms, such as Isomap [Tenenbaum et al., 2000] and Locally Linear Embedding (LLE) [Roweis and Saul, 2000], approach this problem by casting it as an over-constrained convex optimization problem in the low-dimensional space. Unfortunately, much is lost in casting the inherently non-convex problem as a convex problem. The solution to the convex problem can typically be computed rapidly, but the results do not necessarily preserve the distances and angles between neighboring points as well as can be done in low-dimensional space. Algorithms that perform optimization in the high-dimensional space, such as Maximum Variance Unfolding (MVU) [Weinberger et al., 2004], produce better results, but tend to have unreasonably high computational costs.

We make the novel observation that the optimization problem inherent in manifold learning can be solved using graduated optimization. Graduated optimization involves solving a sequence of successively more difficult optimization problems, where the solution to each problem gives a good starting point for the next problem, as illustrated in Figure 3.1. This technique is commonly used with hierarchical pyramid methods for matching objects within images [Burt, 1981]. A related technique called numerical continuation [Wu, 1996] has been used to approximate solutions to parameterized equations in chaotic dynamical systems, molecular conformation, and other areas. To our knowledge, graduated optimization has not yet been recognized as being suitable for addressing the problem of manifold learning. With graduated optimization, if the first optimization problem in the sequence can be solved, and if the solution to each problem falls within the locally-convex region around the solution to the next problem, then it will find the globally optimal solution to the non-convex optimization problem at the end of the sequence.

We present an algorithm for manifold learning called *Manifold Sculpting*, which discovers manifolds through a process of graduated optimization. Manifold Sculpting approaches the optimization problem of manifold learning in a manner that enables it to solve the optimization problem in the original high-dimensional space, while only requiring the computational cost of optimizing in the reduced low-dimensional space. Further, because graduated optimization is robust to local optima, it is not necessary to cast it as an over-constrained convex optimization problem. Instead, Manifold Sculpting directly optimizes to restore the relationships computed between neighboring points. This gives Manifold Sculpting the flexibility to operate using an arbitrary set of distance metrics or other relationship metrics. Additionally, Manifold Sculpting has the ability to benefit from prior knowledge about expected results and the ability to further refine the results from faster manifold learning algorithms. We report results from a variety of experiments which demonstrate that Manifold Sculpting yields results that are typically about an order of magnitude more accurate than state-of-the-art manifold learning algorithms, including Hessian Locally Linear Embedding, and Landmark Maximum Variance Unfolding.

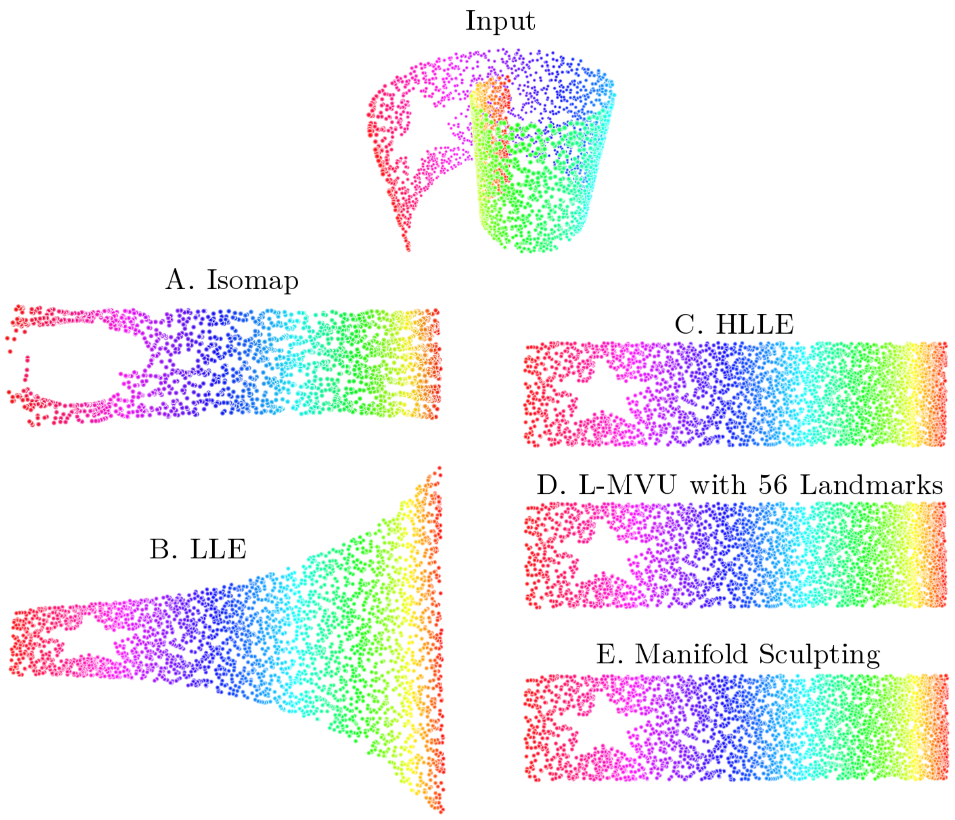


Figure 3.2: Comparison of several manifold learners with a Swiss Roll manifold. Color is used to indicate how points in the results correspond to points on the manifold. Isomap has trouble with sampling holes. LLE has trouble with changes in sample density. HLLE, L-MVU, and Manifold Sculpting all produce very good results with this particular problem. (Results with other problems are presented in Section 3.4.)

Section 3.2 discusses work that has previously been done in manifold learning. Section 3.3 describes the Manifold Sculpting algorithm in detail. Section 3.4 reports the results of a thorough empirical analysis comparing Manifold Sculpting with existing manifold learning algorithms. Finally, Section 3.5 summarizes the contributions of Manifold Sculpting.

3.2 Related Work

Dimensionality reduction has been studied for a long time [Hotelling, 1933], but has only started to become a mainstay of machine learning in the last decade. More algorithms exist than we can mention, but we will attempt to give a summary of the major work that has been done in this field. Early nonlinear dimensionality reduction algorithms, such as Nonlinear Multidimensional Scaling [Shepard and Carroll, 1965] and Nonlinear Mapping [Sammon, 1969], have shown effectiveness, but are unable to handle high nonlinearities in the data. Curvilinear Component Analysis (CCA) [Demartines and Hérault, 1997] uses a neural network technique to solve the manifold embedding, and Curvilinear Distance Analysis (CDA) [Lee et al., 2000] takes it a step further by using distance on the manifold surface as a metric for identifying manifold structure. These algorithms are unfortunately both computationally demanding, and suffer from the problems of local minima.

Isomap [Tenenbaum et al., 2000] uses the same metric as CDA, but solves for the embedding into fewer dimensions using classic multidimensional scaling, which enables it to operate significantly faster. Unfortunately, it still struggles in poorly sampled areas of the manifold. (See Figure 3.2.A.) Locally Linear Embedding (LLE) [Roweis and Saul, 2000] achieves even better speed by using only local vector relationships represented in a sparse matrix. It is more robust to sample holes, but tends to produce quite distorted results when the sample density is non-uniform. (See Figure 3.2.B.) With these algorithms, a flurry of new research in manifold learning began to produce numerous new techniques. L-Isomap is an improvement to the Isomap algorithm that uses landmarks to reduce the amount of necessary computation [de Silva and Tenenbaum, 2002]. Other algorithms include Kernel Principal Component Analysis [Schölkopf et al., 1999], Laplacian Eigenmaps [Belkin and Niyogi, 2001], Manifold Charting [Brand, 2003], Manifold Parzen Windows [Vincent and Bengio, 2003], Hessian LLE [Donoho and Grimes, 2003], and there are many more [Bengio and Monperrus, 2005, Levina and Bickel, 2005, Zhang and Zha, 2006]. Hessian LLE preserves the manifold structure better than the other algorithms but is, unfortunately, very computationally expensive. (See Figure 3.2.C.)

More recently, the Maximum Variance Unfolding (MVU) algorithm has become popular for manifold learning [Weinberger et al., 2004]. This algorithm seeks to maximize variance in the data points while preserving distances and angles in local neighborhoods. It finds the solution to this problem using semidefinite programming. Unfortunately, because it optimizes in the original high-dimensional space, and because of the computational complexity of semidefinite programming, it is too inefficient to operate on large datasets. Landmark Maximum Variance Unfolding (L-MVU) [Weinberger et al., 2005] utilizes randomly chosen landmarks to reduce the computational complexity of MVU. (See Figure 3.2.D.) Although this technique yields somewhat degraded results, it makes the algorithm more suitable for larger problems. Excessive computational complexity, however, is still the most significant drawback of L-MVU.

Several recent manifold learning algorithms have also been presented with specialized capabilities. For example, LGGA [Huang et al., 2009] creates a continuous mapping, such that out-of-sample points can be projected efficiently onto a learned manifold. We show that Manifold Sculpting can achieve a similar capability using a pseudo-incremental technique. S-Isomap [Geng et al., 2005] has the ability to benefit from partial supervision. Manifold Sculpting can also utilize supervision to improve its manifold embedding. TRIMAP [Chen et al., 2010] and the D-C Method [Meng et al., 2008] are manifold learning techniques that specifically seek to preserve class-separability in their projections for classification tasks. Manifold Sculpting is not designed specifically for this application.

The primary difference between other methods that have been presented and Manifold Sculpting is that others use various convex optimization techniques to approximate a solution to the non-convex problem of preserving relationships in local neighborhoods, while the latter seeks to directly solve this non-convex optimization problem with the use of graduated optimization. Manifold Sculpting was first presented in [Gashler et al., 2008b]. In this paper, we present an improved version of the algorithm, demonstrate additional capabilities, and show that it gives better results than modern algorithms. (Figure 3.2.E shows that results with the Swiss Roll manifold are visually similar to those of HLLE or L-MVU. In Section 3.4, we show that an empirical analysis with this and several other problems indicates that the results of Manifold Sculpting tend to be much more accurate than those of other algorithms.)

Table 3.1: A high-level overview of the Manifold Sculpting algorithm.

1	Find the k-nearest neighbors of each point.
2	Compute a set of relationships between neighboring points.
3	Pre-process the data with a faster dimensionality reduction algorithm. (This step is optional.)
4	Do until no improvement is made for 50 iterations:
	a. Scale the data in the non-preserved dimensions by a constant factor σ, where $\sigma < 1$.
	b. Restore the relationships computed in step 2 by adjusting the data points in the first t dimensions.
5	Drop the non-preserved dimensions from the data.

3.3 The Manifold Sculpting Algorithm

An overview of the Manifold Sculpting algorithm is provided in Table 3.1, and detailed pseudo code is provided in Figure 3.5. Let

d	$\equiv$ The original dimensionality of the data.
t	$\equiv$ The number of target dimensions into which the data will be projected.
k	$\equiv$ The number of neighbors used to define a local neighborhood.
$\mathbf{P}$	$\equiv$ The set of all data points represented as vectors in $\Re^d$, such that p_{ij} is the j^{th} dimensional element of the i^{th} point in $\mathbf{P}$.
$\mathbf{N}$	$\equiv$ A $\lvert\mathbf{P}\rvert \times k$ matrix such that n_{ij} is the index of the j^{th} neighbor of point $\mathbf{p}_{i,*}$.
σ	$\equiv$ A constant scaling factor.
η	$\equiv$ The step size (which is dynamically adjusted).

3.3.1 Steps 1 and 2: Compute local relationships

Manifold Sculpting can operate using custom distance/relationship metrics. In our implementation, we use Euclidean distance and local angles. We compute the k-nearest neighbors,

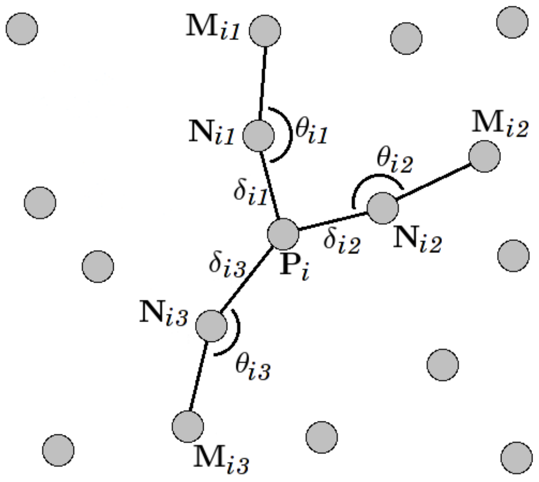


Figure 3.3: δ and θ define the relationships that Manifold Sculpting seeks to preserve in the projection.

$\mathbf{n}_{ij}$, of each point. For each j (where $1 \le j \le k$) we compute the Euclidean distance δ_{ij} between $\mathbf{p}_{i,*}$ and each of its neighbor points. We also compute the angle θ_{ij} formed by the two line segments ($\mathbf{p}_{i,*}$ to point n_{ij}) and (point $\mathbf{n}_{ij}$ to point $\mathbf{m}_{ij}$), where point $\mathbf{m}_{ij}$ is the most co-linear neighbor of point $\mathbf{n}_{ij}$ with $\mathbf{p}_{i,*}$. (See Figure 3.3.) The most co-linear neighbor is the neighbor point that forms the angle closest to π. The values of δ and θ define the relationships that the algorithm will seek to preserve during the transformation. The global average distance between all the neighbors of all points δ_{ave} is also computed so that distances may be normalized.

3.3.2 Step 3: Optionally pre-process the data

The data may optionally be pre-processed with another dimensionality reduction algorithm. Manifold Sculpting will work without this step; however, pre-processing may result in even faster convergence. For example, a fast but imprecise algorithm, such as LLE, may be used to initially unfold the manifold, and then Manifold Sculpting can further refine its results to obtain a better embedding. (This technique is demonstrated in Section 3.4.2.) Even pre-processing with a linear dimensionality reduction algorithm may give some speed benefit.

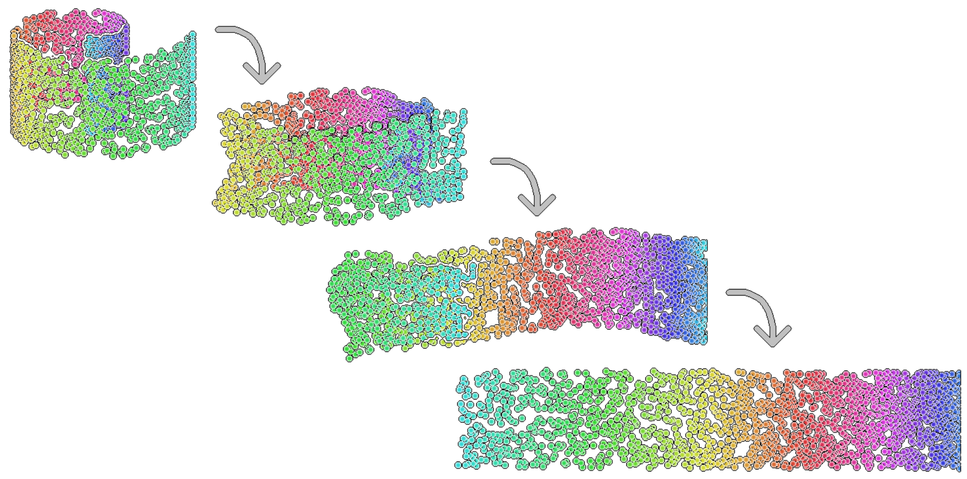


Figure 3.4: A Swiss Roll manifold shown at four stages of the iterative transformation. This experiment was performed with 2000 data points, $k = 14$, $\sigma = 0.99$, and iterations = 300.

For example, Principal Component Analysis (PCA) can be used to rotate the dimensional axes in order to shift the information in the data into the first several dimensions. Manifold Sculpting will then further compact the information by unfolding the non-linear components in the data. Except where otherwise noted, we use PCA to pre-process data in this manner.

Efficient PCA algorithms generally compute only the first few principal components, and simultaneously project away the additional dimensions. In this case, however, it is preferable to rotate the axes to align the data with the first few principal components without projecting away the remaining dimensions, since that will be done later in step 5. The additional information in those dimensions is useful in step 4 for reducing tension in the graduated optimization step. Section 3.6 gives pseudo-code for aligning axes with the first few principal components without projecting away the additional dimensions.

3.3.3 Step 4: Transform the data

The data is iteratively transformed as shown in Figure 3.4. This transformation continues until at least $log(0.01)/log(\sigma)$ iterations are performed, and the sum error has not improved over a window of 50 iterations. The first criterion ensures that most (99%) of the variance is scaled out of the dimensions that will be dropped in the projection. The second criterion allows the algorithm to operate as long as it continues to improve the results. These criteria can be modified to suit the desired quality of results – if precision is important, a larger window may be used; if speed is important, early stopping may be appropriate.

Step 4a: Reduce the variance in non-target dimensions by scaling.

All the values in $\mathbf{P}$ except those in the first t dimensions are scaled down by multiplying by a constant factor σ, where σ is slightly less than 1. This value controls the rate at which the optimization problem is graduated. A conservative value, such as $\sigma = 0.999$, will ensure that the error surface is transformed slowly, so that the global optimum can be followed with precision. A more liberal value, such as $\sigma = 0.99$, can be used to obtain results quickly, with some risk of falling into a local optimum. Except where otherwise indicated, we use the value $\sigma = 0.99$ for all of the experiments in this paper.

To compensate for this down-scaling in the non-preserved dimensions, the values in the first t dimensions are scaled up to keep the average neighbor distance equal to δ_{ave}. As the algorithm iterates, this will shift the variance out of the non-preserved dimensions, and into the preserved dimensions. Thus, when the projection is performed in step 5, very little information will be lost.

Step 4b: Restore original relationships.

Next, the values in the first t dimensions in P are adjusted to recover the relationships that are distorted by scaling in the previous step. This is the optimization phase of graduated optimization. A heuristic error value is used to measure the extent to which the current relationships between neighboring points differ from the original relationships:

$$\epsilon_{p_i} = \sum_{j=0}^{k} w_{ij} \left(\left(\frac{\delta_{ij_0} - \delta_{ij}}{2\delta_{ave}} \right)^2 + \left(\frac{\max(0, \theta_{ij_0} - \theta_{ij})}{\pi} \right)^2 \right) \tag{3.1}$$

where δ_{ij} is the current distance to point n_{ij}, δ_{ij_0} is the original distance to point n_{ij} measured in step 2, θ_{ij} is the current angle, θ_{ij_0} is the original angle measured in step 2.

The denominators are normalizing factors that give each term approximately equal weight. When $\epsilon_{p_i} = 0$, the relationship metrics have been restored to their original values. We adjust the values in the first t dimensions of each point to minimize this error value. Since the equation for the true gradient of the error surface defined by this heuristic is complex, and is $O(d^3)$ to compute, we use the simple hill-climbing technique of adjusting in each dimension in the direction that yields improvement until a local optimum is found. This technique is sufficient to follow the trough of the changing error surface. Pseudo-code for our hill-climbing technique is given in Figure 3.6.

Three performance optimizations can be used in this step to significantly speed convergence:

First, it is observed that the component of distances and angles in the non-preserved dimensions does not change, except that it is scaled by σ in each iteration. These values can be cached (as long as the cached values are scaled by σ at each iteration) such that only the first t dimensions must be evaluated to compute the error heuristic. Since t tends to be a small constant value, this can have a significant impact on runtime performance.

Second, the step size, η, can be dynamically adjusted to keep the number of total steps low. We adjust η after each iteration to keep the total number of steps taken approximately equal to the total number of data points. If the number of steps is less than the number of data points, then $\eta \leftarrow \eta * 0.9$, otherwise $\eta \leftarrow \eta * 1.1$. Experimentally this technique was found to converge significantly faster than using a constant or decaying value for η.

Third, it is observed that the points which have already been adjusted in the current iteration have a more positive influence for guiding the movement of other points to reduce overall error than the points which have not yet been adjusted. Thus, we begin step 4b by starting with a randomly selected point, and we visit each point using a breadth-first traversal ordering. Intuitively this may be analogous to how a person smoothes a crumpled piece of paper by starting at an arbitrary point and smoothing outward. Thus, higher weight is given to the component of the error contributed by neighbors that have already been adjusted in the current iteration, such that $w_{ij} = 1$ if point n_{ij} has not yet been adjusted in this iteration, and $w_{ij} = 10$ if point n_{ij} has been adjusted. Intuitively, a large weight difference will promote faster unfolding, while a smaller weight difference should promote more stability. In our experiments, nearly identical results were obtained using values as low as $w_{ij} = 5$ or as high as $w_{ij} = 20$ for the case where a neighbor has already been adjusted, so we made no attempt to further tune w_{ij} in any of our experiments.

3.3.4 Step 5: Project the data

At this point nearly all of the variance is contained in the first t dimensions of P. The data is projected by simply dropping all but the first t dimensions from the representation of the points.

3.3.5 Graduated Optimization

The optimization technique used by Manifold Sculpting warrants particular consideration. The algorithm relies on a simple hill climber to adjust the location of each point (in step 4b). It cycles through the dimensions, and for each dimension it tries increasing the value and decreasing the value, and it accepts whichever yields improvement (or leaves the point unchanged if neither yields improvement). By itself, this is an unsophisticated optimization technique that is highly susceptible to falling into a local optimum. The problem over which Manifold Sculpting ultimately seeks to optimize, however, is not convex. Thus, an additional component is necessary to ensure that Manifold Sculpting obtains good results.

The key observation of Manifold Sculpting is that after the relationships between neighboring points has been computed, but before any transformation begins, the error value will be zero. No set of values for the data points can produce a lower error, so the system begins in a stable state. The hill climber need not seek a global optimum from a random starting point. Rather, it need only remain in a stable state while the variance is iteratively

```
function manifold_sculpting($\mathbf{P}$)
  1   for $i$ from 0 to $|\mathbf{P}|-1$:
          $\mathbf{n}_i \leftarrow$ the $k$ nearest neighbors of $\mathbf{p}_{i,*}$
  2   for $i$ from 0 to $|\mathbf{P}|-1$:
          for $j$ from 0 to $k-1$:
              $\delta_{ij} \leftarrow \mathbf{P}.\operatorname{distance}(i, n_{ij})$
              $\theta_{ij} \leftarrow \max_{0<l\leq k} \mathbf{P}.\operatorname{angle}(i, n_{ij}, n_{jl})$
              $\mathbf{M}_{ij} \leftarrow \operatorname{argmax}_{0<l\leq k} \mathbf{P}.\operatorname{angle}(i, n_{ij}, n_{jl})$
      $\delta_{ave} \leftarrow$ average_neighbor_distance()
      $\eta \leftarrow \delta_{ave}$
  3   Call align_axes_with_principal_components($\mathbf{P}$)
  4   Until at least ($\log_\sigma 0.01$) iterations, and until no
       improvement is made for 50 iterations, do:
 4a       for $i$ from 0 to $|\mathbf{P}|-1$:
              for $j$ from $t$ to $d-1$:
                  $p_{ij} \leftarrow \sigma p_{ij}$
          while average_neighbor_distance() $< \delta_{ave}$:
              for $i$ from 0 to $|\mathbf{P}|-1$:
                  for $j$ from 0 to $t-1$:
                      $p_{ij} \leftarrow p_{ij}/\sigma$
 4b       $r \leftarrow$ random value, $0 \leq r < |\mathbf{P}|$
          add point $r$ to a queue
          $steps \leftarrow 0$
          $\mathcal{A} \leftarrow$ empty set
          while the queue is not empty, do:
              pop index $i$ from the queue
              if $i \notin \mathcal{A}$:
                  $steps \leftarrow steps +$ adjust_point($\mathbf{p}, \eta$)
                  add each neighbor of $\mathbf{p}_{i,*}$ to the queue
                  add $i \rightarrow \mathcal{A}$
          if $steps \geq |\mathbf{P}|$:
              $\eta \leftarrow \eta * 1.1$
          else
              $\eta \leftarrow \eta * 0.9$
  5   Drop all dimensions $\geq t$
```

Figure 3.5: Pseudo code for the Manifold Sculpting algorithm. Note that pseudo code for the align_axes_with_principal_components function is given in Section 3.6, and pseudo code for the adjust_point function is given in Figure 3.6. A C++ implementation of Manifold Sculpting is available online at http://waffles.sourceforge.net.

```
function adjust_point($\mathbf{p}, \eta$)
 $s \leftarrow 0$
 loop:
     $\epsilon_0 \leftarrow$ compute_error($\mathbf{p}$)
     $j \leftarrow 0$
     for $i$ from 0 to $t - 1$:
         $p_i \leftarrow p_i + \eta$
         if compute_error($\mathbf{p}$) $< \epsilon_0$:
             $j \leftarrow j + 1$
         else
             $p_i \leftarrow p_i - 2\eta$
             if compute_error($\mathbf{p}$) $< \epsilon_0$:
                 $j \leftarrow j + 1$
             else
                 $p_i \leftarrow p_i + \eta$
     if $j = 0$:
         return $s$
     $s \leftarrow s + 1$
```

Figure 3.6: Pseudo code for the adjust_point function, where compute_error is Equation 3.1. This is a convex hill climbing algorithm that moves a point to a locally-optimal position, and returns the number of steps required to get there.

scaled out of the higher dimensions. Thus, by gradually transforming the problem from these initial conditions, a simple hill-climbing algorithm can be sufficient to follow the optimum.

Gradient-based optimization may be comparable to rolling a ball down a hill and hoping that it finds its way to a good local optimum. The optimization technique used by Manifold Sculpting, on the other hand, is more analogous to a ball following at the feet of a person walking across a trampoline. It begins adjacent to the person's feet, and seeks to remain by them as the person moves slowly across the trampoline. As long as the ball never deviates very far from the person's feet, the topology of the rest of the trampoline is irrelevant to the final destination of the ball because the trampoline will always be locally convex around the person's feet.

To understand why this approach tends to be effective, let us first consider the hypothetical case where the manifold represented by the original dataset is topologically close to the optimally transformed data. In other words, suppose the optimally transformed data with all variance in the first t dimensions can be obtained through a continuous transformation. Further, suppose that there exists such a continuous transformation that would also preserve the relationships in local neighborhoods at all times during the transformation. It follows that there will be a globally optimal embedding (with an error of zero) at any time during the transformation. Further, because the transformation is topologically continuous, that embedding will follow a continuous path through the error space, beginning at the point that represents the original data, and ending with the optimally transformed data. At any given time, this embedding, which is known to be optimal with respect to the error surface at that time, will be found at the exact bottom of a locally-convex basin. Thus, if the continuous transform is approximated with small enough values for σ (the scaling rate) and η (the step size), then the system can be certain to finally arrive at the globally optimal solution. This is not a proof that Manifold Sculpting will always yield optimal results, however, because there is no guarantee that there exists such a continuous transformation. On the other hand, it is not a pathological situation either. The swiss roll manifold, for example, can be trivially shown to meet all of these conditions.

In cases where neighborhood relationships must be temporarily violated in order to "unfold" a manifold, it is often still reasonable to expect that the lowest valley in the error surface will follow a continuous path through the error space, even if that valley does not remain at zero. Non-continuous jumps in the error space correspond to "tearing" of a manifold structure to instantly separate previously adjacent parts. Intuitively, this is not typically a desirable behavior. Even in cases where the best transformation requires such jumps in the error space, it is still likely that it will yet arrive at a good local optimum. It should do no worse than techniques that simply use convex optimization. Further, necessary rips in the manifold structure are not likely to be frequent events. If the system becomes temporarily separated from the global optimum, it may yet be able to find the optimum again as the error surface continues to change. The global optimum is the only optimum that is certain to remain a local optimum during the entire transformation.

3.3.6 Parameter Tuning

Although this algorithm has several parameters that could, in theory, be tuned to obtain better results with a particular problem, we have not found that it is typically necessary to do so. In the experiments reported in this paper, we have only adjusted k (the number of neighbors), t (the number of target dimensions), and σ (the scaling rate). The parameters k and t are common in all of the algorithms with which we compare. The scaling rate, however, is unique to Manifold Sculpting. For most problems, rapid convergence can be obtained using the value $\sigma = 0.99$, but for complex manifold topologies, a slower scaling rate, such as $\sigma = 0.999$, may be necessary to give the manifold more iterations to unfold.

3.3.7 Estimating Intrinsic Dimensionality

The error heuristic used by Manifold Sculpting is an indicator of how well neighborhood structure has been preserved. In cases where the intrinsic dimensionality of a problem is not known *a priori*, the variance may be scaled out of the higher dimensions one at a time. The error heuristic will then indicate how well the manifold structure is preserved into each number of dimensions. In contrast with eigenvalues, this error heuristic will indicate the component of non-linear variance in the manifold. When the error begins to climb rapidly, the intrinsic dimensionality of the manifold has been subceeded. This feature is particularly convenient for analysis in which the intrinsic dimensionality must be determined. When the intrinsic dimensionality is known, however, as is the case with the experiments in this paper, it is preferable for efficiency to scale all of the extra dimensions simultaneously.

3.4 Empirical Analysis

We tested the properties of Manifold Sculpting with a wide variety of experiments and a range of manifolds. Section 3.4.1 reports empirical measurements of accuracy using toy-problems that have known ideal results. These results indicate that Manifold Sculpting is more accurate than Isomap, LLE, HLLE, and L-MVU with these problems. Section 3.4.2 demonstrates the capabilities of Manifold Sculpting using several image-based manifolds. Section 3.4.3 reports on an experiment with document-based manifolds. Section 3.4.4 discusses using partial supervision, and training Manifold Sculpting in a pseudo-incremental manner.

3.4.1 Accuracy

Figure 3.2 (page 18) shows that Manifold Sculpting appears visually to produce results of higher quality than LLE and Isomap with the Swiss Roll manifold, a common visual test for manifold learning algorithms. Quantitative analysis shows that it also yields more precise results than HLLE and L-MVU. Since the actual structure of this manifold is known prior to using any manifold learner, we can use this prior information to quantitatively measure the accuracy of each algorithm.

We define a Swiss Roll in 3D space with n points (x_i, y_i, z_i) where $0 \leq i < n$ as follows: Let $t = 8i/n + 2$, $x_i = t\sin(t)$, y_i is a random number $-6 \leq y_i < 6$, and $z_i = t\cos(t)$. In 2D manifold coordinates, the corresponding target points are (u_i, v_i), such that $u_i = \frac{\sinh^{-1}(t)+t\sqrt{t^2+1}}{2}$ and $v_i = y_i$. To emphasize the effect of poorly-sampled areas, we also removed samples that fell within a star-shaped region on the manifold as shown in Figure 3.2.

We created a Swiss Roll with 2000 data points and reduced the dimensionality to 2 with each of four algorithms. We tested how well the output of each algorithm aligned with the target output values, (u_i, v_i), by measuring the mean squared distance from each point to its corresponding target value. Since there is no guarantee how the results would be oriented, we used the affine transformation that most closely aligned the results with the expected

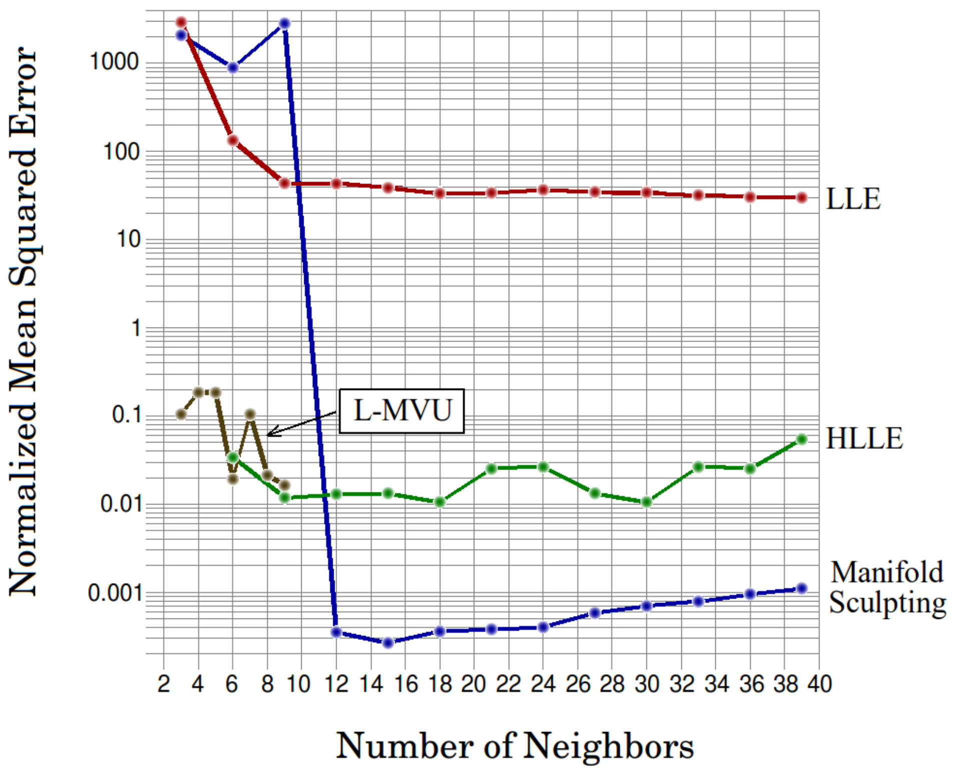


Figure 3.7: The mean squared error of four algorithms for a Swiss Roll manifold using a varying number of neighbors k. The vertical axis is shown on a logarithmic scale. The excessive memory requirement of L-MVU and HLLE limited the range of neighbors over which we were able to test these algorithms, but L-MVU did very well with few neighbors. Manifold Sculpting yielded the most accurate results when at least 12 neighbors were used.

results before measuring the mean squared distance. We then normalized the mean squared error by dividing by λ, where λ is the square of the average distance between each point in the target dataset and its nearest neighbor. Thus, a normalized mean squared error larger than 1 would probably indicate that the results are significantly distorted, since the average deviation of a point from its ideal location is more than the distance between neighboring points.

Figure 3.7 shows the normalized mean squared distance between each transformed point and its expected value. Results are shown with a varying number of neighbors k. Both axes are shown on a logarithmic scale. With L-MVU, 44 landmarks ($\approx \sqrt{n}$) were used. The resource requirements of L-MVU became unreasonable after 9 neighbors, as they did

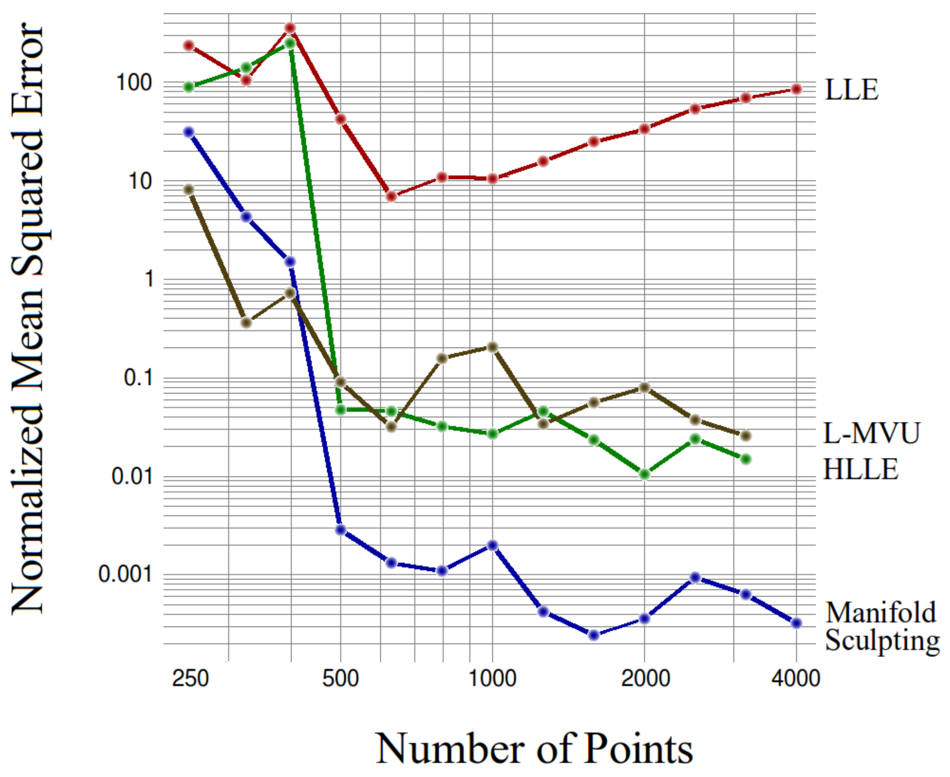


Figure 3.8: The mean squared error of four algorithms with a Swiss Roll manifold using a varying number of points. Both axes are shown on a logarithmic scale.

for HLLE after 48 neighbors. LLE and Manifold Sculpting could easily handle many more neighbors, but neighbors begin to cut across manifold boundaries at that point. Isomap yielded very poor results and is not shown. Manifold Sculpting did not yield good results until at least 12 neighbors were used. This may indicate that L-MVU is a better choice when so few samples are available that many neighbors would be unreasonable. With this problem, L-MVU, HLLE, and Manifold Sculpting can all produce results that are very close to the ideal results. Proportionally, however, the results from Manifold Sculpting are precise by more than an order of magnitude over the next-best algorithm.

We repeated the experiment with the Swiss Roll manifold using a varying number of points to sample the manifold. Figure 3.8 shows the results of this experiment. 18 neighbors were used for LLE, HLLE, and Manifold Sculpting because these algorithms all did well with this value in the previous experiment. For L-MVU, 6 neighbors were used to keep the resource

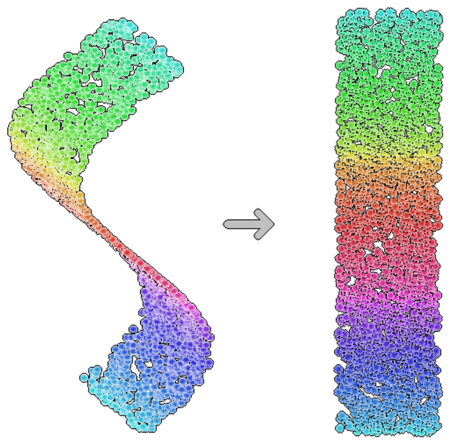


Figure 3.9: A visualization of an S-Curve manifold.

requirements manageable, and because it did well with this value in the previous experiment. We used the square root of the number of points (rounded down) for the number of landmarks. The memory requirements for HLLE and L-MVU became unreasonable after 3175 points. HLLE and L-MVU yielded very good results with this problem. When the manifold was sampled with at least 500 points, however, Manifold Sculpting produced proportionally more accurate results than the other algorithms. L-MVU yielded the best results when the manifold was sampled with fewer points, but none of the algorithms yielded very good results with fewer than 500 sample points. HLLE, L-MVU, and Manifold Sculpting all appear to exhibit the trend of producing better accuracy as the sample density is increased.

To verify that these results were not peculiar to the Swiss Roll manifold, we repeated this experiment using an S-Curve manifold as depicted in Figure 3.9. This manifold was selected because we could also compute the ideal results for it by integrating to find the distance over its surface. We defined the S-Curve points in 3D space with n points (x_i, y_i, z_i) where $0 \leq i < n$ as follows: Let $t = \frac{(2.2i-0.1)\pi}{n}$, $x_i = t$, $y_i = sin(t)$, and z_i is a random number $0 \leq z_i < 2$. In 2D manifold coordinates, the corresponding target points are (u_i, v_i), such that $u_i = \int_0^t \left(\sqrt{\cos^2(w) + 1}\right) dw$ and $v_i = z_i$. We measured accuracy in the same manner described in the previous two experiments. These results are shown in Figure 3.10.

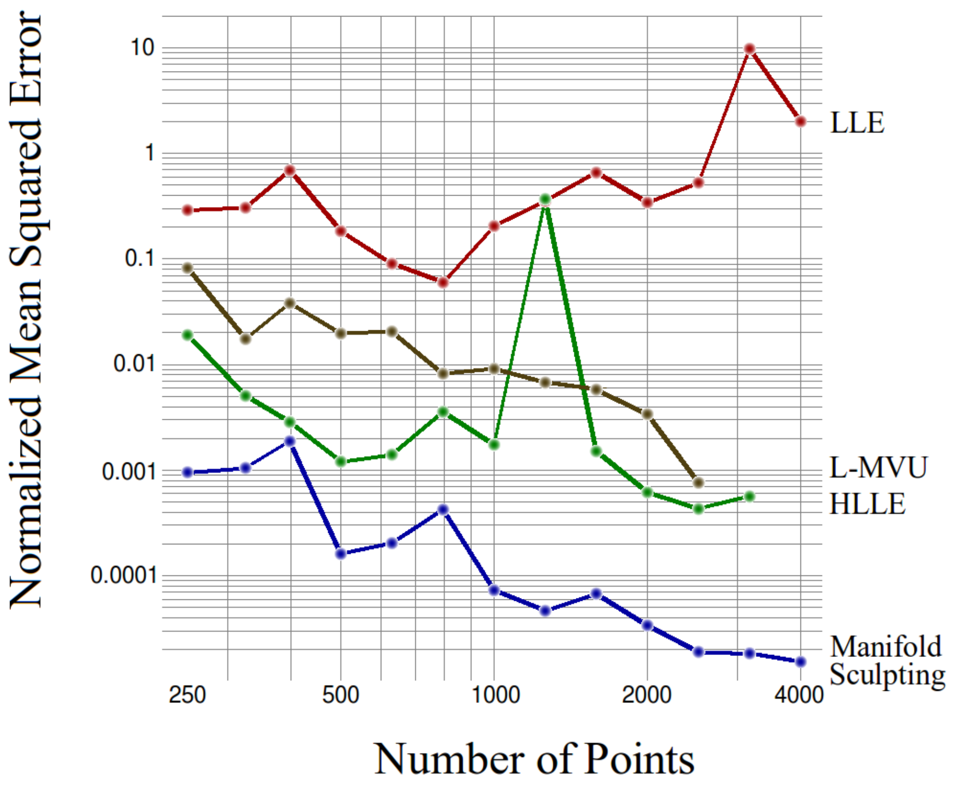


Figure 3.10: The mean squared error of four algorithms for an S-Curve manifold using a varying number of points. Manifold Sculpting consistently outperformed all other algorithms on this problem, and is more than an order of magnitude better than HLLE, which is the closest competitor. Both axes are shown on a logarithmic scale.

Again, results are not shown for L-MVU or HLLE with very large numbers of points due to the demanding resource requirements of these algorithms. Consistent with the previous experiment, L-MVU, HLLE, and Manifold Sculpting all produced very good results with this simple manifold. The trends exhibited in these results were similar to those from the Swiss Roll manifold, with Manifold Sculpting producing results that were proportionally better.

A test was also performed with an Entwined Spirals manifold as shown in Figure 3.11. In this case, Isomap produced the most accurate results, even though it consistently had the poorest results for all manifolds with an intrinsic dimensionality greater than 1. (See Figure 3.12.) This can be attributed to the nature of the Isomap algorithm. In cases where the manifold has an intrinsic dimensionality of exactly 1, a path from neighbor to neighbor

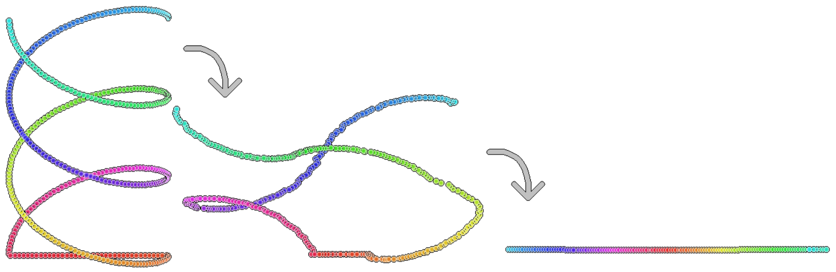


Figure 3.11: A visualization of an Entwined Spirals manifold.

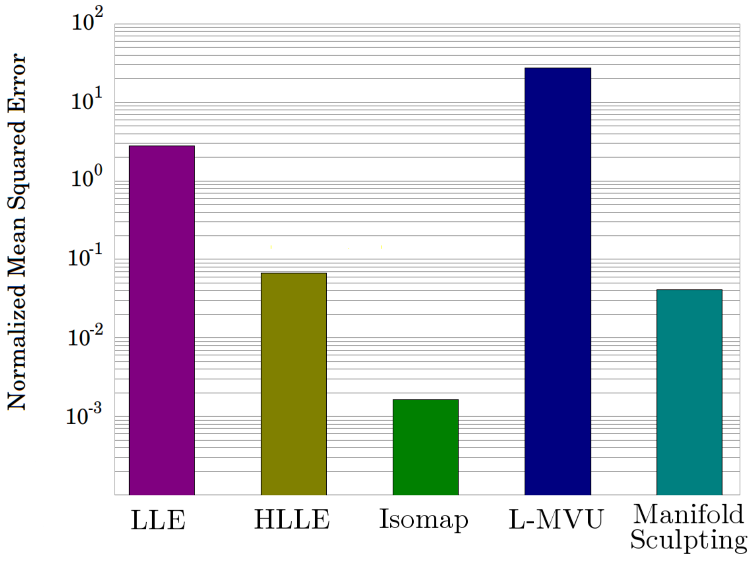


Figure 3.12: Mean squared error for five algorithms with an Entwined Spirals manifold. Isomap does very well when the intrinsic dimensionality is exactly 1.

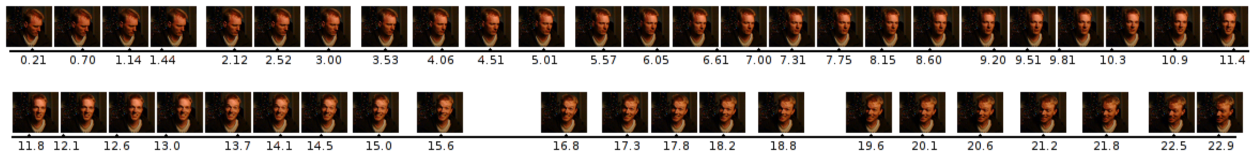


Figure 3.13: Images of a face reduced by Manifold Sculpting into a single dimension. The values are shown here on two wrapped lines in order to fit the page. The original image is shown above each point.

provides an accurate estimate of isolinear distance. Thus an algorithm that seeks to globally optimize isolinear distances will be less susceptible to the noise from cutting across local

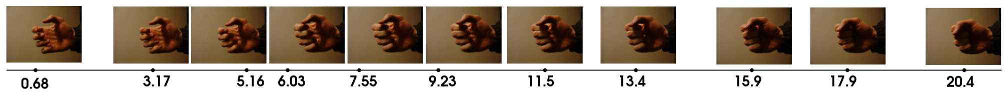


Figure 3.14: Images of a hand reduced to a single dimension. The original image is shown above each point.

corners. When the intrinsic dimensionality is higher than 1, however, paths that follow from neighbor to neighbor produce a zig-zag pattern that introduces excessive noise into the isolinear distance measurement. In these cases, preserving local neighborhood relationships with precision yields better overall results than globally optimizing an error-prone metric. Consistent with this intuition, Isomap yielded very accurate results in our other experiments, reported hereafter, that involved a manifold with a single intrinsic dimension, and yielded the poorest results with experiments in which the intrinsic dimensionality was larger than one.

3.4.2 Image-based manifolds

Many unsupervised learning problems do not have a corresponding set of ideal results. The Swiss Roll and S-Curve manifolds are useful for quantitative analysis because expected results can be computed *a priori*, but real-world applications are likely to involve many more than just three dimensions. We therefore performed several experiments to demonstrate that Manifold Sculpting is also accurate with problems that involve much larger initial dimensionality. Figure 3.13 shows several frames from a video sequence of a person turning his head while gradually smiling. Each image was encoded as a vector of $1,634$ pixel intensity values. No single pixel contained enough information to characterize a frame according to the high-level concept of facial position, but this concept was effectively encoded in multi-dimensional space. This data was then reduced to a single dimension. (Results are shown on two separate lines in order to fit the page.) The one preserved dimension could then characterize each frame according to the high-level concept that was previously encoded in many dimensions. The dot below each image corresponds to the single-dimensional value in the preserved dimension for that image. In this case, the ordering of every frame was consistent with the ordering in the

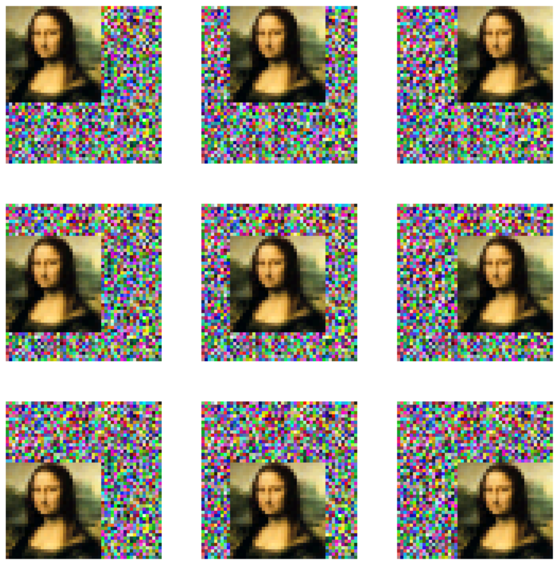


Figure 3.15: A dataset was generated by translating an image over a background of noise. Nine representative images are shown. Results from several algorithms using this dataset are shown in Figure 3.16.

video sequence. Because the ideal results with this problem are not known, it is not possible to compute the accuracy of these results. We therefore did not compare with other algorithms using this problem, but the correctness of these results is somewhat visually apparent.

Figure 3.14 shows eleven images of a hand. These images were encoded as multi-dimensional vectors in the same manner. The high-level concept of "hand openness" was preserved into a single dimension. In addition to showing that Manifold Sculpting can work with real-world data, this experiment also shows the robustness of the algorithm to poorly sampled manifolds because these eleven images were the only images used for this experiment. Again, the ideal results with this problem are not known, so no accuracy measurement is given.

Another experiment involves a manifold that was generated by translating a picture over a background of random noise as shown in Figure 3.15. This figure shows a sample of 9 images. The manifold was sampled with 625 images, each encoded as a vector of 2,304 pixel intensity values. Because two variables (horizontal position and vertical position) were used to generate the dataset, the data can be interpreted as sampling from a manifold with an

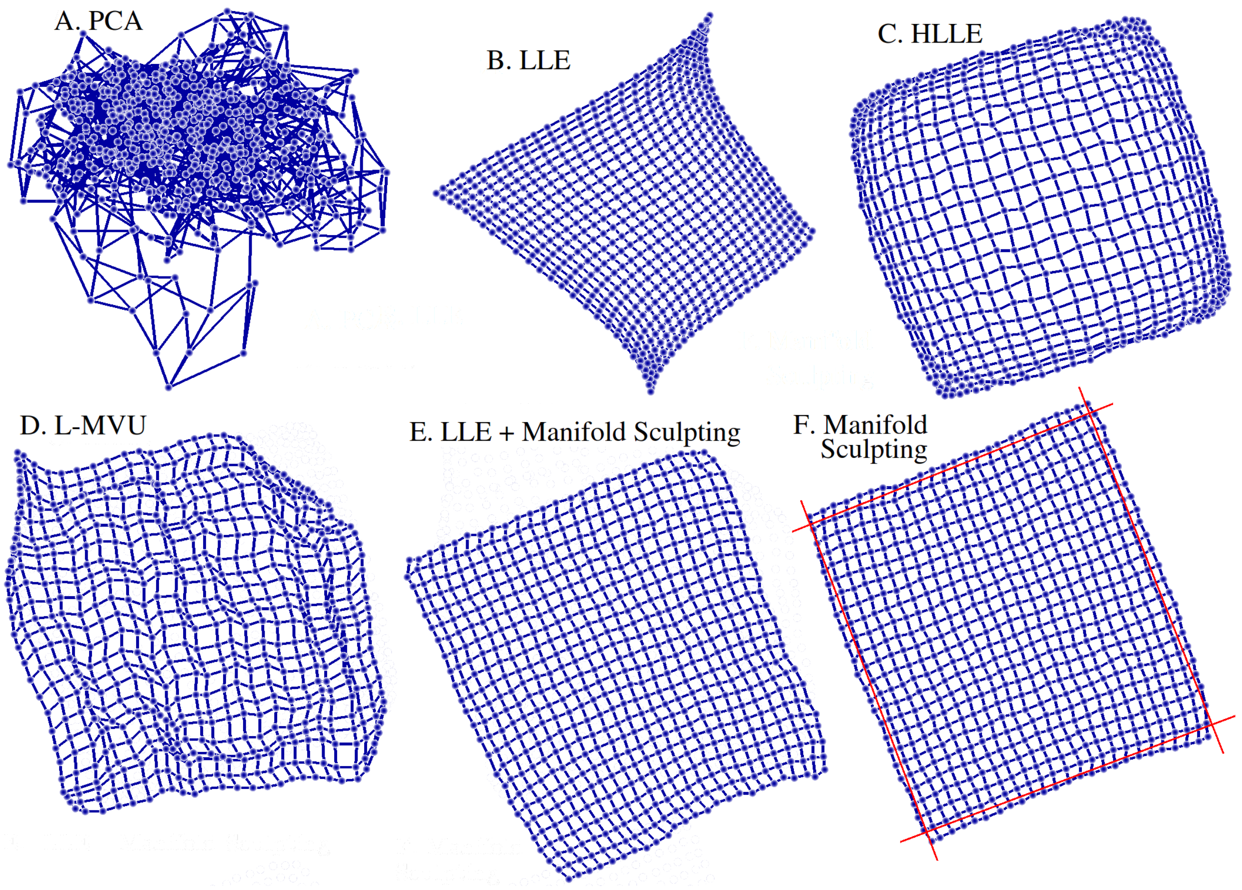


Figure 3.16: A comparison of results with a manifold generated by translating an image over a background of noise. 8 neighbors were used with HLLE, L-MVU, and Manifold Sculpting. 4 neighbors were used with LLE because it produced better results than with 8 neighbors. 49 landmarks were used with L-MVU. Results for Manifold Sculpting are shown with LLE pre-processing using the default scaling rate $\sigma = 0.99$, and with only PCA pre-processing using a slower scaling rate of $\sigma = 0.999$. The results from Manifold Sculpting are nearly linearly separable.

intrinsic dimensionality of two in a space with an extrinsic dimensionality of 2,304. Because the background is random, the average distance between neighboring points in the input space should be somewhat uniform, therefore the ideal reduced-dimensionality result can be expected to occupy a space very close to square in shape. We, therefore, use this as a basis for empirically measuring the quality of results. Quantitative measurements with this problem may be a better indicator of the strengths of a manifold learning algorithm than the Swiss Roll or S-curve manifolds because: 1) This problem involves reduction from high-dimensional space, and 2) To our knowledge, highly accurate results have not yet been obtained with this problem.

Figure 3.16 shows a comparison of results from various algorithms on this problem. For increased visibility of the inherent structure, each vertex is shown connected with the four nearest neighbors in the input space. Results are shown with PCA to demonstrate how very non-linear this manifold is (see Figure 3.16.A). We tested the other algorithms with 4 and 8 neighbors (because the points lie on a grid-like structure), and report the best results for each algorithm. LLE did better with 4 neighbors. The other algorithms did better with 8 neighbors.

To demonstrate the ability of Manifold Sculpting to benefit from the results of other dimensionality reduction algorithms, we substituted LLE for the pre-processing (step 3 of the Manifold Sculpting algorithm). These results, shown in Figure 3.16.E, were obtained rapidly using the default scaling rate of $\sigma = 0.99$. The best results, however, were obtained using a slower scaling rate ($\sigma = 0.999$). These results are shown in Figure 3.16.F. In this case, we used the default PCA pre-processing. With this problem, it is necessary to use a slower scaling rate when PCA is used for pre-processing so that the manifold has sufficient time to unfold.

We empirically measured the quality of the results obtained in this experiment by comparing results with points distributed evenly over a perfect square. These results are shown in Figure 3.17. LLE and HLLE do poorly because their results tend to exhibit global distortions, which have a more significant impact on the mean squared error. L-MVU achieves a normalized mean squared error less than 1. It exhibits a mixture of both global and local distortions. The mean-squared-error of the result from Manifold Sculpting is more than an order of magnitude smaller than that of L-MVU.

Another observation that can be made from the results shown in Figure 3.16.F is that the results are approximately linearly separable. For example, if it were desirable to classify these images into two classes such that class A contained all images in which the picture of the Mona Lisa is adjacent to the top of the image, and class B contained all other images, this could be done using the results from Manifold Sculpting with a single linear division boundary. Since this dataset was designed to have predictable results, these classes would only have pathological applications, but this demonstrates the significant potential of Manifold Sculpting to create separability of intrinsic concepts from otherwise complex data.

With many problems, however, distances in observation space are not uniformly scaled in relation to the intrinsic variables. For example, Figure 3.18A shows the results of using Manifold Sculpting to reduce the dimensionality of a manifold generated by sliding a window over a larger picture of the Mona Lisa. This result is not square because some parts of the image exhibit different gradients than other parts. When the window slides over a region with a larger gradient, a bigger distance is measured. In order to obtain results that more correctly represent the intrinsic variables in this problem, we used a custom distance-metric that normalized distances in local neighborhoods, such that each neighborhood represents a uniform amount of total distance. This result is shown in Figure 3.18B. These values are a better representation of the intrinsic values in this problem because they are less biased by the irrelevant gradient in the images. This experiment used 1296 images, each represented as a vector of 3675 continuous pixel-values.

In addition to supporting custom relationship metrics, Manifold Sculpting is also flexible regarding the representation of intrinsic points. For example, if a subset of supervised points is available, these points can be clamped with their supervised values while Manifold

Sculpting operates, and Manifold Sculpting will naturally find values for the other points that fit well in relation to the supervised points. This capability is useful for at least 3 potential applications: 1- It can be used to improve runtime performance. With several problems we were able to measure as much as a 50% speedup when as few as 5% of the points had supervised values. Since the supervised points are clamped with their known values, they tend to act as a force that pulls the other points directly toward their final destinations. 2- It can be used to align intrinsic values with the dimensional axes, or to give the distances between them meaning with respect to a particular unit. This might be useful, for example, if the intrinsic values are used to estimate the state of some system. 3- It can be used to facilitate pseudo-incremental manifold learning. If, for example, points arrive from a stream, pseudo-incremental manifold learning may be useful to update the reduced-dimensional estimate for all of the points that have yet arrived. This is done in two steps: First, Manifold Sculpting is applied with values clamped to all the points that have known values. This rapidly computes values for the new incoming points. Second, Manifold Sculpting is applied again with all points un-clamped, but starting in the location of their reduced-dimensional values. This enables all of the points to be updated in consequence of the new information, but also incurs very little computational cost since all points begin already in nearly-optimal locations.

3.4.3 Document Manifolds

The utility of manifold learning algorithms for image processing applications has recently been recognized, but this is certainly not the only field that deals with multi-dimensional data. The Vector Space Model [Raghavan and Wong, 1986], for example, is commonly used in the field of Information Retrieval to characterize web documents. Each web page is represented as a large vector of term weights. The number of dimensions in the vector corresponds to the number of unique word stems (about 57,000 in English), and the values in these dimensions correspond to a term weight computed from the number of occurrences of the

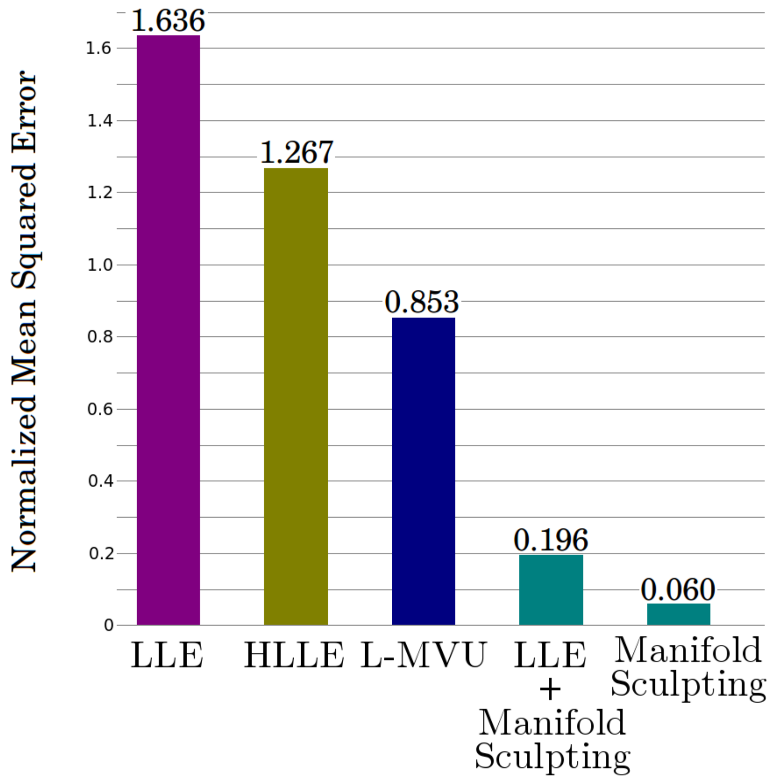


Figure 3.17: An empirical measurement of the error of the results shown in Figure 3.16. Error was computed as the smallest mean squared error from points distributed evenly over a square. An affine transformation was found to align results as closely as possible with the square before error was measured. Results from Manifold Sculpting are more than an order of magnitude better than the next closest competitor.

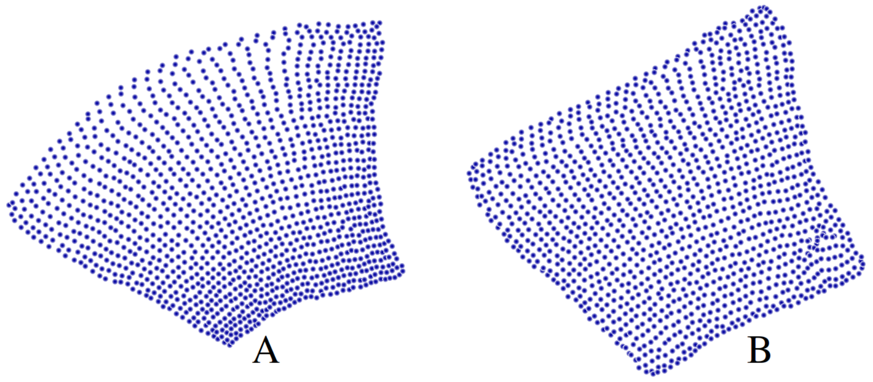


Figure 3.18: A. Manifold Sculpting was used to reduce the dimensionality of a manifold generated from a collection of 1296 images, each represented as a vector of 3675 continuous pixel values. These images were generated by sliding a window over a larger picture of the Mona Lisa. This result is not square because the sliding window creates a bigger change in input distance when sliding over regions with a bigger gradient. B. A custom distance-metric was used to normalize distance such that each neighborhood represents a uniform amount of total distance. This result better represents the intrinsic variables of this problem.

term in a document. Search queries can be evaluated by finding the documents whose vector has the closest angle with the vector of the query. This representation bears some striking resemblances to the pixel representation used for processing images, so it seems likely that similar results could be obtained by applying manifold learning to this field.

To test this hypothesis, we implemented a simple application for refining the results obtained from a *Google* search. A typical search often yields many thousands of documents, but users rarely have patience to look at more than the first few. Our application downloads the first 100 documents, removes stop words, stems each term with the Porter stemming algorithm [Porter, 1980], and represents the documents with the Vector Space Model. Next it uses Manifold Sculpting to reduce the dimensionality of the vectors to a single dimension. It then divides this dimension into two halves at the point that minimizes the sum variance of the two halves. Finally it extracts three terms to represent each of the two clusters by summing the vectors in the cluster and picking the three terms with the highest total weight. In theory, these two groups of terms should reflect the most significant range of context found in the query results, and a user should be able to refine his or her query by selecting which of the two halves is closer to the intended meaning of the query.

As an example, a query for the term "speaker" yielded for one set of terms "box", "amp", and "off", and for the other set "hit", "baseball", and "bat". The first interpretation of the term "speaker" probably comes from references to audio devices. Prior to performing this experiment we did not know that this term had anything to do with baseball, but a search for the refined query "speaker baseball" yields many documents with information about Tris Speaker who was elected to the baseball hall of fame in 1937. Not all queries yielded such distinctly separated results, but this is an area with potential for further research.

3.4.4 Semi-supervision

Manifold learning and clustering have many things in common. Clustering collections of multidimensional vectors such as images, for example, is more effective when manifolds in the data are taken into account [Breitenbach and Grudic, 2005]. Manifold learning and clustering are both unsupervised operations. Unlike clustering however, which groups vectors into a discrete number of buckets, manifold learning arranges them into a discrete number of continuous spectra. In some sense, clustering is to manifold learning what classification is to regression. It seems intuitive, therefore, that techniques which benefit clustering algorithms may have a corresponding counterpart for manifold learning algorithms. Clustering algorithms can be greatly enhanced with partial supervision [Demiriz et al., 1999]. Likewise, a small modification to the Manifold Sculpting algorithm makes it possible to perform semi-supervised manifold learning.

Semi-supervised clustering involves a subset of data points for which classification values or hints about those values are provided. Semi-supervised manifold learning likewise requires final values or estimated final values to be provided for a subset of the data points. During scaling iterations (step 4 of the Manifold Sculpting algorithm), these points are clamped to their supervised values. The unsupervised points are free to move, but will

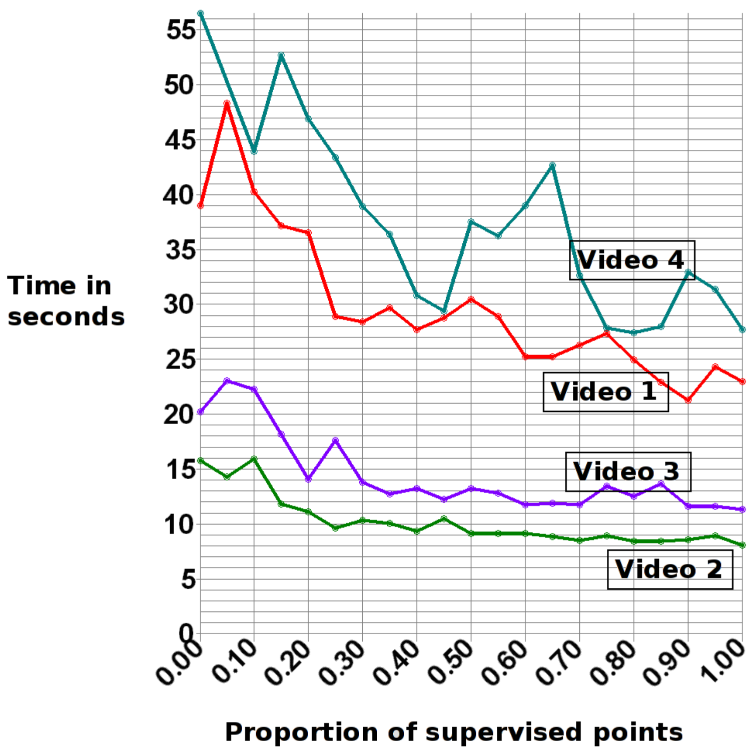


Figure 3.19: Manifold learning is faster when more points are supervised. (When most of the points are supervised, the only significant cost is the neighbor-finding step of the algorithm.)

be influenced by their neighbors, some of which may be supervised. This results in more efficient and potentially more accurate manifold learning. Figure 3.19 shows the amount of time required to learn the intrinsically one-dimensional manifold for each of the four video sequences with a varying percentage of supervised points. It can be observed in these results that the first 20% to 50% of supervised points tend to produce noticeable improvements in speed, but additional supervised points tend to make little difference.

In some cases data may not be available all at once. In such cases it may be desirable to learn a manifold as the data becomes available from a stream [Law et al., 2004]. Pseudo-incremental learning is naturally achieved with semi-supervised manifold learning. The following two-step process is followed when new data points become available: First, the old points are clamped to their known reduced-dimensionality values and the new points are allowed to settle. Second, all points are unclamped, and the entire dataset is allowed to settle. The first step is very fast because most of the points are supervised. The second step is also fast because the manifold is already unfolded, and all of the points are likely to be very close to their final states. Significant computation is only required when new data causes the overall structure of the manifold to change.

Often, semi-supervised clustering is performed by having human experts classify a subset of the available data. When real values are required, however, it may only be reasonable to ask human experts to provide vague estimates. For example, humans might be able to easily sort a collection of images according to relevance to a particular topic. The sorted images could be assigned sequential output values, and these values could be used as estimates for the points in manifold coordinates. However, precise values may be difficult to obtain initially. Fortunately, even estimated output values can be useful in semi-supervised manifold learning. This benefit is exploited by first clamping estimated values to guide the unfolding of the manifold, and then unclamping all of the points during later iterations so that more precise refinements can be made to the data.

3.5 Conclusions

A significant contribution of this paper is the observation that the optimization step of manifold learning is suitable to be solved using graduated optimization, a technique that can rapidly find the global optimum with many otherwise-difficult optimization problems. We demonstrated this by presenting a manifold learning algorithm called Manifold Sculpting, which uses graduated optimization to find a manifold embedded in high-dimensional space.

The collection of experiments reported in this paper indicate that Manifold Sculpting yields more accurate results than other well-known manifold learning algorithms for most problems. We collected empirical measurements using a Swiss Roll manifold with varying sample densities and with varying numbers of neighbors. We also tested with an S-Curve manifold, an Entwined Spirals manifold, a manifold generated by translating an image over a background of noise, and a few other manifolds. Isomap tends to be very strong when the intrinsic dimensionality of a problem is exactly 1, and L-MVU does well when very few sample points are available, but in all other cases, Manifold Sculpting yielded the most accurate results, and is typically at least an order of magnitude more accurate than the closest competitor algorithm.

The results produced by Manifold Sculpting are robust to parameter choices, except for the scaling rate. We showed that the default scaling rate ($\sigma = 0.99$) works well with most problems, but that a slower scaling rate will yield good results with more complex problems. We also demonstrated that Manifold Sculpting can benefit by pre-processing the data with other dimensionality reduction algorithms. In many cases, this enables accurate results to be obtained rapidly, without resorting to the use of a slower scaling rate.

3.6 PCA for graduated optimization

This appendix section provides pseudo code for the align_axes_with_principal_components function. This performs the same function as the axis rotation step of principal component analysis, except that it preserves data in all dimensions while only aligning with the first t principal components. This differs from regular PCA, which aligns data with the first few principal components and then throws out all values in the remaining dimensions. With Manifold Sculpting, it may be preferable to preserve these values because they represent some component of the distances between points. Thus, this algorithm may be used instead of regular PCA for preprocessing the data prior to applying Manifold Sculpting with the slight advantage that this technique guarantees not to affect any of the distances in local neighborhoods. This pseudo code is given in Figure 3.20.

```
function align_axes_with_principal_components($\mathbf{P}$)
 $\mu \leftarrow$ compute_mean($\mathbf{P}$)
 for each $\mathbf{p}_{i,*} \in \mathbf{P}$:
     $\mathbf{p}_{i,*} \leftarrow \mathbf{p}_{i,*} - \mu$
 $\mathbf{Q} \leftarrow$ copy_of($\mathbf{P}$)
 $\mathbf{G} \leftarrow \{\hat{\imath}, \hat{\jmath}, \hat{k}, \ldots\}$ such that $|\mathbf{G}| = t$
 for $k$ from 0 to $t-1$
     $\mathbf{c} \leftarrow$ a random vector of size $t$
     do 20 times:
         $\mathbf{v} \leftarrow$ zero vector of size $t$
         for each $\mathbf{q}_i \in \mathbf{Q}$:
             $\mathbf{v} \leftarrow \mathbf{v} + (\mathbf{q}_i \cdot \mathbf{c})\mathbf{q}_i$
         $\mathbf{c} \leftarrow \frac{\mathbf{v}}{|\mathbf{v}|}$
     for each $\mathbf{q}_i \in \mathbf{Q}$:
         $\mathbf{q}_i \leftarrow \mathbf{q}_i - (\mathbf{c} \cdot \mathbf{q}_i)\mathbf{c}$
     $\mathbf{a} \leftarrow \mathbf{G}_k$
     $\mathbf{b} \leftarrow \frac{\mathbf{c}-(\mathbf{a}\cdot\mathbf{c})\mathbf{a}}{|\mathbf{c}-(\mathbf{a}\cdot\mathbf{c})\mathbf{a}|}$
     $\phi \leftarrow \arctan(\frac{\mathbf{b}\cdot\mathbf{c}}{\mathbf{a}\cdot\mathbf{c}})$
     for $j$ from $k$ to $t-1$
         $u \leftarrow \mathbf{a} \cdot \mathbf{G}_j$
         $v \leftarrow \mathbf{b} \cdot \mathbf{G}_j$
         $\mathbf{G}_j \leftarrow \mathbf{G}_j - u\mathbf{a}$
         $\mathbf{G}_j \leftarrow \mathbf{G}_j - v\mathbf{b}$
         $r \leftarrow \sqrt{u^2 + v^2}$
         $\theta \leftarrow \arctan(\frac{v}{u})$
         $u \leftarrow r\cos(\theta + \phi)$
         $v \leftarrow r\sin(\theta + \phi)$
         $\mathbf{G}_j \leftarrow \mathbf{G}_j + u\mathbf{a}$
         $\mathbf{G}_j \leftarrow \mathbf{G}_j + v\mathbf{b}$
 for each $\mathbf{p}_{i,*} \in \mathbf{P}$:
     for $j$ from 0 to $t-1$:
         $p_{ij} \leftarrow \mathbf{p}_{i,*} \cdot \mathbf{G}_j + \mu_j$
```

Figure 3.20: Pseudo code for the align_axes_with_principal_components function. This performs the same function as the axis rotation step of principal component analysis, except this algorithm only aligns with the first $|D_{preserved}|$ principal components while preserving data in all dimensions.

Chapter 4

Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous

Abstract: Using decision trees that split on randomly selected attributes is one way to increase the diversity within an ensemble of decision trees. Another approach increases diversity by combining multiple tree algorithms. The random forest approach has become popular because it is simple and yields good results with common datasets. We present a technique that combines heterogeneous tree algorithms and contrast it with homogeneous forest algorithms. Our results indicate that random forests do poorly when faced with irrelevant attributes, while our heterogeneous technique handles them robustly. Further, we show that large ensembles of random trees are more susceptible to diminishing returns than our technique. We are able to obtain better results across a large number of common datasets with a significantly smaller ensemble.

4.1 Introduction

Ensembles offer a simple yet effective technique for obtaining increased levels of predictive accuracy by combining the predictions of many different learning algorithm instances [Hansen and Salamon, 1990, Opitz and Maclin, 1999, Dietterich, 2000, Polikar, 2006]. However, such improvements are predicated upon there existing some form of diversity among the elements of the ensemble [Sollich and Krogh, 1996, Kuncheva and Whitaker, 2003]. Indeed, if every instance in the ensemble behaves nearly the same way, little is achieved by combining their predictions.

Decision trees are particularly well-suited for ensembles because they are fast and unstable. Hence, it is often possible to create synergy within decision tree ensembles or forests. A popular technique for promoting variance in decision tree forests is to use trees that randomly choose on which attributes to divide the data [Ho, 1995]. In sufficiently large ensembles, this technique can yield better accuracy than standard entropy-reducing decision trees on many datasets because it creates more variance within the models. Breiman showed that bagging is particularly effective with forests of random decision trees [Breiman, 2001]. On the downside, however, our results indicate that random trees yield poor results on data with many irrelevant attributes.

In this paper, we contrast the random forest algorithm with another ensemble technique that combines multiple tree algorithms using cross-validation selection. We show that this technique both has a higher ceiling of diminishing returns and is more robust to irrelevant features than homogeneous tree ensembles. Heterogeneity in our ensembles is achieved through a combination of entropy-reducing decision trees, which build axis aligned decision boundaries, and a new class of decision trees, known as mean margins decision trees (MMDT), which build oblique decision boundaries.

The paper is organized as follows. Section 4.2 briefly reviews significant related work. In Section 4.3, we describe the mean margins decision tree learning algorithm and the hybrid cross-validation decision tree learning algorithm which may be obtained by combining the mean margins decision tree learning algorithm with the standard entropy-reducing decision tree learning algorithm. Section 4.4 presents a thorough analysis of the resulting heterogeneous ensemble learning algorithm. Finally, Section 4.5 concludes the paper.

4.2 Related Work

The MMDT algorithm we introduce here uses linear combinations of inputs to define the decision boundaries of its induced model trees. Such trees were first discussed in [Breiman et al., 1984] and later implemented in a number of algorithms, such as Multivariate Decision Trees [Brodley and Utgoff, 1995], Oblique Decision Trees [Murthy et al., 1994], and Perceptron Decision Trees [Utgoff, 1989].

Linear combination trees offer significant flexibility over trees that only divide data with axis-aligned boundaries. Unfortunately, this flexibility tends to be a hindrance more than a benefit. Training a perceptron tree, for example, involves optimization in a very non-convex heuristic space. Further, there is a strong tendency for perceptron trees to use their extreme flexibility to overfit the training data. MMDT, on the other hand, is asymptotically as efficient as the well-known entropy-reducing decision tree learning algorithm. It is also parameterless and tends to produce good results with many datasets.

Although a complete review is outside the scope of this paper, much of the research involving ensemble methods is clearly relevant to our work (e.g., see [Dietterich, 2000, Brown et al., 2005, Freund and Schapire, 1995, Dietterich, 2002]). Of particular interest is work on diversity in ensembles. The need for diversity in ensemble is well known and has been the object of many studies. Many techniques have been proposed from bagging [Breiman, 1996] to stacking [Wolpert, 1992] to mixture of experts [Jacobs et al., 1991] to random forests [Breiman, 2001], to COD-based approaches [Peterson and Martinez, 2005], to name only a few. A recent survey of techniques for creating diversity in ensemble is in [Brown et al., 2005]. We use the term *homogeneous* to refer to techniques that use a single algorithm and achieve diversity through some form of variability in the data (e.g., randomization), and we use the term *heterogeneous* to refer to techniques that achieve diversity through the use of multiple algorithms. A thorough comparison of randomization-based decision tree ensemble methods is in [Banfield et al., 2007]. Our approach is heterogeneous and is compared with one well-known homogeneous approach, namely random forests.

4.3 Mean Margins Decision Tree Learning

We first describe a new decision tree learning algorithm called Mean Margins Decision Tree (MMDT) learning, which builds oblique decision boundaries. The MMDT algorithm is designed to be simple, efficient, and free of parameters. It is, therefore, well-suited for use in ensembles. Suppose we have a set of patterns, P, for a binary classification problem such that P_T is the subset of patterns of class $true$, and P_F is the subset of patterns of class $false$. Further, suppose all patterns are vectors of real values.

```
function build_tree($P$)
    $\vec{\mu}, \vec{\nu} \leftarrow$ choose_decision_boundary($P$)
    $P_{left}, P_{right} \leftarrow$ divide_data($\vec{\mu}, \vec{\nu}, P$)
    if $|P_{left}| == 0$
        return new LeafNode($P_{right}$)
    if $|P_{right}| == 0$
        return new LeafNode($P_{left}$)
    $node_{left} \leftarrow$ build_tree($P_{left}$)
    $node_{right} \leftarrow$ build_tree($P_{right}$)
    return new InteriorNode($node_{left}, node_{right}$)

function choose_decision_boundary($P$)
    $\vec{\mu}_F \leftarrow \frac{1}{|P_F|} \sum_{\vec{p} \in P_F} \vec{p}$
    $\vec{\mu}_T \leftarrow \frac{1}{|P_T|} \sum_{\vec{p} \in P_T} \vec{p}$
    $\vec{\mu} \leftarrow \frac{\vec{\mu}_F + \vec{\mu}_T}{2}$
    $\vec{\nu} \leftarrow \vec{\mu}_T - \vec{\mu}_F$
    return $\vec{\mu}, \vec{\nu}$

function divide_data($\vec{\mu}, \vec{\nu}, P$)
    $P_{left} \leftarrow \{\}$
    $P_{right} \leftarrow \{\}$
    for each $\vec{p} \in P$
        if $(\vec{p} - \vec{\mu}) \cdot \vec{\nu} \geq 0$
            $P_{right} \leftarrow P_{right} + \vec{p}$
        else
            $P_{left} \leftarrow P_{left} + \vec{p}$
    return $P_{left}, P_{right}$
```

Figure 4.1: MMDT Learning Algorithm

At a high level, an MMDT is constructed in a manner very similar to that of any other decision tree, as shown in Figure 4.1. The difference is that the MMDT algorithm chooses decision boundaries in the form of linear combinations of inputs that maximize the margins between the means of P_T and P_F, as illustrated in Figure 4.2.

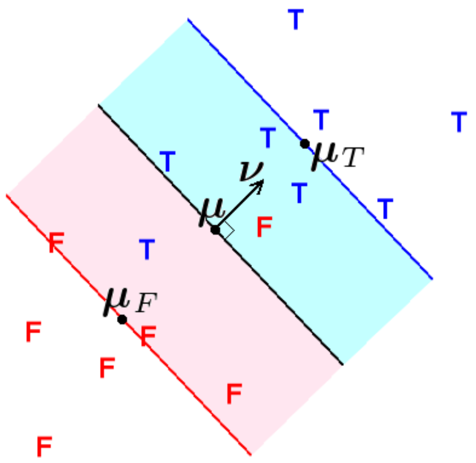


Figure 4.2: Margin Maximization

Of course, not all classification tasks have exactly two classes and only real-valued inputs. However, the MMDT algorithm is easily extended to handle nominal attributes, and any number of classes. To handle nominal attributes, we represent each value as an orthogonal dimension. For example, suppose some nominal attribute ranges over the values {red, green, blue}, and some pattern contains the value $v =$ red. We would represent this value with three real values $< 1, 0, 0 >$. These values may be thought of as a categorical distribution of confidence over the nominal values. (Note that this technique also naturally provides a mechanism for handling missing nominal values: just assign equal confidences to each value, e.g., $< 0.33, 0.33, 0.33 >$.) To convert back to a nominal value, just find the value with the maximum confidence.

To handle more than two classes, we use the following technique. If class labels are nominal, they are also converted to real vectors as per the above procedure, such that L is the set of real vector labels (one for each pattern in P). Each time before *choose_decision_boundary* is called, we first divide P into P_T and P_F in the following manner:

```
function compute_1st_principal_component($L$)
    for $i = 1$ to $d$
        $\vec{\nu}_{Li} \leftarrow random\_standard\_normal()$
    $\vec{\nu}_L \leftarrow \frac{\vec{\nu}_L}{|\vec{\nu}_L|}$
    do 10 times
        $\vec{\alpha} \leftarrow 0_d$
        for each $\vec{l} \in L$
            $\vec{\alpha} \leftarrow \vec{\alpha} + ((\vec{l} - \vec{\mu}_L) \cdot \vec{\nu}_L)(\vec{l} - \vec{\mu}_L)$
        $\vec{\nu}_L \leftarrow \frac{\vec{\alpha}}{|\vec{\alpha}|}$
```

Figure 4.3: Computing 1st Principal Component

1. Compute the mean $\vec{\mu}_L$ and first principal component $\vec{\nu}_L$ of L.
2. For each $\vec{l} \in L$, if $(\vec{l} - \vec{\mu}_L) \cdot \vec{\nu}_L \geq 0$ then P_T contains patterns with label $\vec{l}$, otherwise P_F contains patterns with label $\vec{l}$.

Thus, the MMDT algorithm can easily work with multiple classes, including continuous labels. For completeness, the pseudocode, derived from [Roweis, 1998], to quickly compute the first principal component about the mean of L in d dimensions is shown in Figure 4.3. In some rare cases, more iterations may be required to obtain a precise estimate of the first principal component, but for this algorithm an imprecise estimate will work just fine, so ten iterations are sufficient.

As an illustration of MMDT's performance in complex environments, we design the following simple interpolation task. Of course, better techniques for image interpolation exist. The purpose here is only to assist an intuition of the workings of MMDT. We create a training set with one pattern per pixel from a small 20x20 pixel image. We then evaluate at sub-pixel positions to interpolate a 160x160 image. We compare bagged MMDT with a random forest of standard entropy-reducing decision trees (ERDTs). The results are shown in Figure 4.4. Note how bagged MMDT is better able to follow non-axis-aligned contours.

As it turns out, the MMDT algorithm is not as effective overall as ERDT learning for common classification tasks, but MMDT tends to perform well in many cases where ERDT

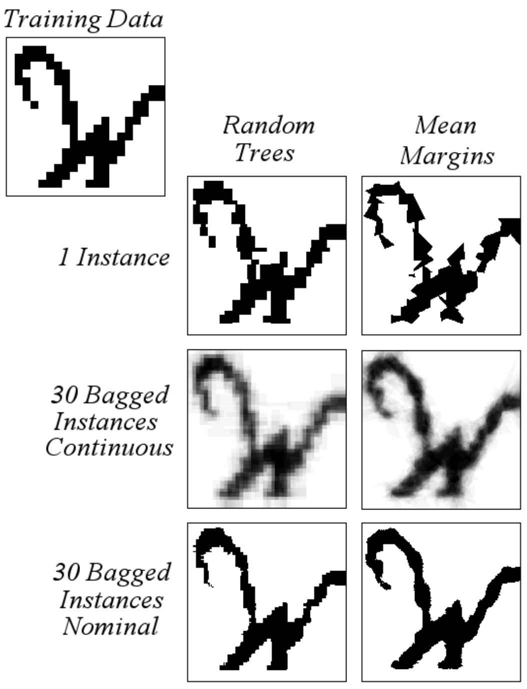


Figure 4.4: MMDT vs RDT Interpolation

does poorly. On a set of 43 common datasets from the UCI repository [Asuncion and Newman, 2007], ERDT achieves higher predictive accuracy (measured by 5x2 cross-validation) than MMDT on almost two thirds (26 out of 43) of the datasets. However, MMDT appears to cover an important deficiency in the remaining one third. MMDT computes mean values in order to choose its decision boundaries. Mean values can be estimated with more accuracy when there is plenty of data. It seems intuitive, therefore, that the MMDT algorithm might do well with datasets that densely sample their input space.

To test this notion, we compute the sample density of each dataset as the product of the arity of each attribute divided by the number of patterns. Since there is no concept of arity with continuous values, we use a value of 5 for continuous attributes, which is close to the average arity of nominal values in the datasets we consider. In a pairwise comparison between bagged MMDT (size 100) and bagged ERDT (size 100) across only the densest half of the datasets, MMDT performs better than ERDT on 45.5% (10 out of 22) of the datasets. On the densest quarter, MMDT does best on 72.7% (8 out of 11) of the datasets, and on the densest eighth, it also does best on 83.3% (5 out of 6) of the datasets. Hence, sample density does seem to characterize a significant portion of the strength of MMDT. There also remain 7 out of 21 datasets (33.3%) among the sparsest half of the datasets on which bagged MMDT outperforms bagged ERDT.

4.4 Decision Tree Ensemble Learning

We will refer to a bagged ensemble of 100 random decision tree (RDT) instances as "100×RDT", to a bagged ensemble of 100 ERDT instances as "100×ERDT", and likewise for other algorithms.

We first compare the predictive accuracy of three bagged ensembles: 100×RDT, 100×ERDT, and 100×MMDT. We measure predictive accuracy on 43 common datasets from the UCI repository [Asuncion and Newman, 2007] using 5x2 cross-validation (5x2CV). Our choice of 5x2CV, rather than the somewhat more popular 10-fold cross-validation, is motivated by recent results which suggest that 5x2CV yields lower type II error than 10-fold cross-validation [Demsar, 2006]. In this experiment, each ensemble is homogeneous in that it contains multiple instances of just one algorithm. The results are shown in Table 4.1. The column "Baseline" corresponds to a majority learner that chooses the most common class. It is shown in order to contrast the effectiveness of the various ensemble techniques. 100×RDT is the most accurate with the most (17) datasets. RDTs are particularly effective at creating diversity within the ensemble, so this result emphasizes the importance of having model diversity. 100×ERDT does best on 13 datases, while 100×MMDT does best on 12 datasets. One dataset has no clear winner.

Given the complementary nature of the strengths of ERDT and MMDT as discussed above, it would seem that performance could be further improved by building heterogeneous ensembles. Rather than combining several ERDTs and MMDTs, we first design the following simple, cross-validation-based decision tree learning algorithm, which we refer to as CVDT (Cross-Validation Decision Tree).

1. Perform 1x2CV on the training set with ERDT
2. Perform 1x2CV on the training set with MMDT
3. Select the algorithm that performed best and train on full training set

We then create heterogeneous ensembles of CVDTs through bagging. When one of the two algorithms (ERDT or MMDT) is clearly better than the other for a particular problem, this is equivalent to a bagged ensemble of that model. When both algorithms achieve similar accuracy, the bagged ensemble will contain a mixture of both algorithms in proportion to the number of times that each achieved better accuracy during cross-validation.

In and of itself, building a heterogeneous ensemble is not that novel, and although training a CVDT requires more computation than training an ERDT or an MMDT, this cost is only required at training time. Evaluation with a CVDT is as efficient as the model that it selects. Furthermore, we will show shortly that, in the context of ensemble learning, significantly smaller ensembles of CVDTs may be used, that achieve higher accuracy and better tolerance to noise than much larger ensembles of ERDTs or random forests, with an overall smaller computational footprint.

We compare ensembles of ERDTs and RDTs with ensembles of bagged CVDTs. A simple analogy motivates this design. A bagged ensemble of ERDTs may be analogous to a panel of expert medical doctors that all graduated from the same university. An ensemble of RDTs may be analogous to a panel of novices that dropped out of medical schools from all over the world. Even though each novice may have less talent than any one of the experts, the diversity in this group may enable them to produce a better combined diagnosis. This may explain why ensembles of RDTs can outperform ensembles of ERDTs. It would seem, therefore, that an ideal panel of medical doctors would contain both a significant amount of diversity *and* expert talent. We seek this balance by using a bagged ensemble of CVDTs. Each CVDT contains only algorithms that build their model with deliberate divisions, but diversity is also enhanced (in addition to the diversity injected as part of the bagging ensemble technique) by the utilization of more than one algorithm.

Here, we consider bagged ensembles of 1,000 ERDTs and 1,000 RDTs with bagged ensembles of only 100 CVDTs. The choice of 1,000 for ensembles of ERDTs and RDTs is rather standard (e.g., see [Banfield et al., 2007]). The results are shown in Table 4.2.

These results indicate that a bagged ensemble of CVDTs has a higher ceiling of diminishing returns than much larger ensembles of RDTs or ERDTs. Furthermore, despite having one tenth the size, the much smaller ensemble of 100×CVDT still yields somewhat higher accuracy on average, and wins outright over both $1,000\times$ERDT and $1,000\times$RDT on 15 of the 43 datastets. On the 16 datasets for which 100×CVDT looses out to both competitors, the loss is usually rather insignificant for at least one of them. Again, these results are obtained at a much lower computational cost since 100×CVDT builds only 500 models (200 ERDTs, 200 MMDTs and 100 CVDTs) rather than the 1,000 required by the other approaches.

In addition to accuracy, we look at how well our proposed ensemble technique handles irrelevant attributes. This property is often ignored in the analysis of many algorithms because popular collections of data tend to contain only attributes that have a fairly significant degree of relevance to the output class. Irrelevant attributes, however, are becoming more and more prevalent as the ease with which data can be collected gives rise to a “let us collect everything we can and worry about its value later” kind of attitude in many machine learning and data mining applications.

For purposes of experimentation, irrelevant attributes are not difficult to generate. We inject varying numbers of attributes containing Gaussian noise into several common datasets. For each dataset, we measure the predictive accuracy of 100×RDT, 100×ERDT, 100×MMDT, and 100×CVDT. Figure 4.5 shows results with the vowel dataset. Other datasets yield very similar trends, so only vowel is shown here as a representative. Note that the horizontal axis is shown on a logarithmic scale, so the right side of the chart represents a broader domain than the left side.

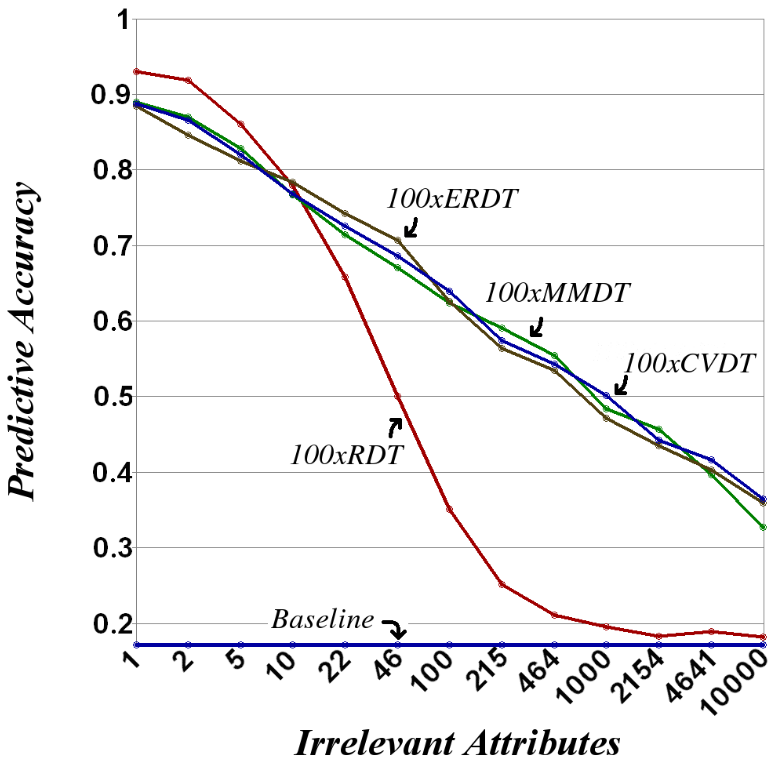


Figure 4.5: Irrelevant Attributes on Vowel

Both 100×ERDT and 100×MMDT handle irrelevant attributes very well. Both algorithms exhibit a nearly linear decrease in accuracy with an exponential increase in the number of irrelevant attributes. It follows, as expected, that the CVDT algorithm, which selects between these two algorithms, also handles irrelevant attributes well. The accuracy of 100×RDT, on the other hand, begins to degrade very quickly after the majority of attributes are irrelevant with respect to class labels.

A common justification for randomly selecting decision boundaries is that this creates models with more variance, and if it randomly decides to split on an irrelevant feature, it may still split on a relevant feature deeper in the tree. As the number of irrelevant features becomes large, however, it constructs models with less and less total relevant information. Consequently, algorithms that identify relevant decision boundaries tend to handle irrelevant attributes better. We, therefore, suggest that utilizing a diversity of algorithms, such as the one proposed here, is a better technique for inducing variance within an ensemble.

4.5 Conclusion

Although RDT may be somewhat effective at producing desirable model variance within an ensemble, heterogeneous ensembles that select from among multiple models can outperform such homogeneous ensembles. Furthermore, ensembles of RDT are not robust to irrelevant attributes.

The MMDT algorithm introduced here is intuitive, efficient, simple to implement and parameterless. It is, therefore, well-suited for use in ensembles. Although, by itself, MMDT is often less accurate than ERDT, MMDT tends to do well with a different set of problems than ERDT. Using cross-validation selection between ERDT and MMDT creates a particularly powerful model. Our results demonstrate that very small ensembles of such cross-validation decision trees (100 vs. 1,000) can outperform very large homogeneous ensembles of RDT both in terms of accuracy and tolerance to irrelevant attributes.

Dataset	***Baseline***	***100x RDT***	***100x ERDT***	***100x MMDT***
colic	0.663	**0.684**	0.637	0.655
balance-scale	0.453	0.846	0.816	**0.877**
segment	0.130	0.966	**0.973**	0.845
breast-cancer	0.703	**0.714**	0.667	0.711
anneal	0.762	0.762	0.762	**0.848**
diabetes	0.651	0.749	**0.758**	0.707
primary-tumor	0.248	**0.419**	0.412	0.413
breast-w	0.655	**0.970**	0.961	0.959
soybean	0.117	0.868	0.810	**0.914**
hepatitis	0.794	**0.831**	0.822	0.775
kropt	0.162	0.630	0.535	**0.652**
badges2	0.714	0.993	**1.000**	0.824
nursery	0.330	0.961	0.971	**0.981**
waveform-5000	0.333	0.825	**0.836**	0.829
heart-c	0.525	**0.829**	0.777	0.583
vowel	0.167	**0.931**	0.874	0.921
zoo	0.406	**0.931**	0.883	0.915
iris	0.289	0.948	0.941	**0.956**
kr-vs-kp	0.522	0.962	**0.989**	0.973
labor	0.583	**0.864**	0.850	0.625
titanic	0.677	0.782	**0.785**	0.784
audiology	0.243	0.615	0.501	**0.717**
credit-a	0.555	**0.865**	0.824	0.648
splice	0.519	0.765	0.934	**0.936**
vehicle	0.240	0.724	**0.746**	0.579
glass	0.534	**0.825**	0.817	0.621
ionosphere	0.641	0.920	0.913	**0.931**
mushroom	0.518	1.000	1.000	1.000
hypothyroid	0.923	0.957	**0.967**	0.923
cars	0.700	0.851	0.906	**0.941**
wine	0.370	0.982	**0.949**	0.674
lymphoma	0.479	0.633	**0.725**	0.604
lenses	0.625	0.608	0.675	**0.758**
heart-statlog	0.527	**0.821**	0.813	0.601
autos	0.327	**0.716**	0.640	0.364
spambase	0.606	**0.949**	0.944	0.727
credit-g	0.700	0.720	**0.723**	0.612
vote	0.614	0.945	**0.956**	0.947
heart-h	0.639	**0.811**	0.793	0.639
lymph	0.547	**0.826**	0.770	0.764
sick	0.939	0.964	**0.971**	0.939
colon	0.645	0.703	0.752	**0.816**
sonar	0.480	**0.781**	0.780	0.773

Table 4.1: Homogeneous Ensembles

Dataset	***Baseline***	***1000x ERDT***	***1000x RDT***	***100x CVDT***	***Improvement Over ERDT***	***Improvement Over RDT***
titanic	0.677	0.788	0.787	0.785	-0.461%	-0.323%
nursery	0.330	0.973	0.965	0.973	-0.019%	0.746%
lenses	0.625	0.725	0.717	0.725	0.000%	1.163%
cars	0.700	0.896	0.856	0.910	1.641%	6.398%
balance-scale	0.453	0.819	0.852	0.870	6.214%	2.064%
iris	0.289	0.948	0.948	0.959	1.125%	1.125%
kropt	0.162	0.537	0.653	0.556	3.586%	-14.757%
vote	0.614	0.958	0.949	0.953	-0.528%	0.438%
diabetes	0.651	0.756	0.754	0.752	-0.551%	-0.345%
primary-tumor	0.248	0.418	0.418	0.417	-0.289%	-0.289%
zoo	0.406	0.891	0.901	0.911	2.183%	1.084%
breast-cancer	0.703	0.657	0.710	0.709	7.987%	-0.197%
badges2	0.714	1.000	0.989	1.000	0.000%	1.100%
breast-w	0.655	0.961	0.970	0.960	-0.091%	-1.003%
glass	0.534	0.834	0.836	0.831	-0.431%	-0.593%
heart-c	0.525	0.759	0.827	0.767	1.038%	-7.262%
heart-h	0.639	0.802	0.816	0.798	-0.509%	-2.250%
vowel	0.167	0.898	0.933	0.919	2.408%	-1.430%
hepatitis	0.794	0.818	0.848	0.812	-0.800%	-4.251%
credit-a	0.555	0.834	0.869	0.833	-0.104%	-4.136%
lymph	0.547	0.764	0.826	0.792	3.717%	-4.092%
heart-statlog	0.527	0.816	0.830	0.810	-0.726%	-2.321%
wine	0.370	0.936	0.984	0.951	1.561%	-3.425%
labor	0.583	0.846	0.873	0.818	-3.292%	-6.373%
kr-vs-kp	0.522	0.990	0.968	0.991	0.101%	2.425%
hypothyroid	0.923	0.970	0.955	0.971	0.022%	1.621%
sick	0.939	0.971	0.964	0.971	0.027%	0.770%
anneal	0.762	0.762	0.762	0.831	9.094%	9.094%
credit-g	0.700	0.726	0.724	0.712	-1.928%	-1.575%
vehicle	0.240	0.746	0.727	0.736	-1.331%	1.202%
segment	0.130	0.974	0.964	0.974	-0.071%	0.988%
sonar	0.480	0.793	0.804	0.795	0.242%	-1.077%
soybean	0.117	0.822	0.875	0.911	10.796%	4.047%
colic	0.663	0.643	0.685	0.638	-0.845%	-6.905%
autos	0.327	0.670	0.728	0.642	-4.211%	-11.786%
audiology	0.243	0.552	0.619	0.737	33.494%	19.170%
ionosphere	0.641	0.922	0.929	0.932	0.994%	0.248%
spambase	0.606	0.943	0.950	0.942	-0.065%	-0.837%
splice	0.519	0.930	0.743	0.939	0.937%	26.322%
mushroom	0.518	1.000	1.000	1.000	-0.022%	-0.030%
waveform-5000	0.333	0.839	0.838	0.830	-1.025%	-0.917%
colon	0.645	0.781	0.661	0.787	0.826%	19.024%
lymphoma	0.479	0.727	0.648	0.713	-2.006%	9.968%
Average					**1.597%**	**0.763%**

Table 4.2: Heterogeneous Ensemble vs. Homogeneous Ensembles

Chapter 5

Robust Manifold Learning With CycleCut

Abstract: Many manifold learning algorithms utilize graphs of local neighborhoods to estimate manifold topology. When neighborhood connections short-circuit between geodesically distant regions of the manifold, poor results are obtained due to the compromises that the manifold learner must make to satisfy the erroneous criteria. Also, existing manifold learning algorithms have difficulty unfolding manifolds with toroidal intrinsic variables without introducing significant distortions to local neighborhoods. An algorithm called *CycleCut* is presented which prepares data for manifold learning by removing short-circuit connections, and by severing toroidal connections in a manifold.

5.1 Introduction

In the last decade, manifold learning has become a significant component of machine learning, data mining, vision, and robotics. In machine learning and data mining, manifold learning can be used to reduce dimensionality, which otherwise presents a significant challenge for many algorithms. In vision and robotics, manifold learning is used to reduce a sequence of images to a corresponding sequence of "intrinsic variables", which can be used to estimate the state from which the images were obtained.

Many manifold learning algorithms, including Locally Linear Embedding, Laplacian Eigenmap, Hessian LLE, Local Tangent Space Alignment, Maximum Variance Unfolding, Manifold Sculpting, and others, rely on graphs of local neighborhoods to estimate manifold topology. These algorithms seek to "unfold" a collection of samples from the manifold to occupy fewer dimensions, while preserving distances (or other relationships) between points in local neighborhoods. The quality of the results obtained from these algorithms is limited by the quality of the neighborhood graphs. If the local neighborhoods contain connections that short-circuit across manifold boundaries, the manifold learner will seek a compromise embedding that satisfies each neighborhood relationship with varying degree, including the short-circuit connection. Such compromise solutions constitute poor results.

We present an algorithm called *CycleCut*, which identifies and "cuts" the neighborhood connections that short-circuit across the manifold. In contrast with existing algorithms, CycleCut efficiently finds a minimal set of connections for removal such that there are no topological holes represented in the manifold. It guarantees not to segment the graph, and under common conditions (which we define in Section 5.3.1), it also guarantees to remove all short-circuiting connections. Even when these conditions do not hold, it tends to exhibit desirable behavior for improving the results of manifold learning. This paper is layed out as follows: Section 5.2 reviews related work. Section 5.3 presents the CycleCut algorithm. Section 5.4 gives an analysis of this algorithm, including a complexity analysis, and empirical results. Section 5.5 concludes by discussing future work.

5.2 Related work

It has long been known that manifold learning algorithms have difficulty when local neighborhood connections short-circuit (or shortcut) across the manifold [Balasubramanian and Schwartz, 2002, Varini et al., 2006]. A simple solution is to use smaller local neighborhoods, but this risks breaking connectivity, and often reduces the quality of results from manifold learning.

A better approach to mitigate this problem involves adaptively selecting local neighborhoods [Wei et al., 2008]. This can make the neighborhood graphs more robust to difficult topologies, but it cannot handle the case where the closest neighbor of a point cuts across the manifold. An approach called manifold denoising [Hein and Maier, 2006] has been used to reduce noise in the data, and thus decrease the risk of shortcut connections. This technique, however, does directly seek to remove shortcut connections, and does not provide any guarantees.

Perhaps the most effective method currently used for identifying shortcut connections is based on Edge Betweenness Centrality (EBC) [Brandes, 2001]. EBC is a metric that counts the number of times that the pair-wise shortest paths between every pair of points pass over each edge. Because shortcut connections join geodesically distant regions of the manifold (by definition), they tend to have higher EBC values than other edges [Cukierski and Foran, 2008]. The drawbacks of this approach are that it sometimes removes edges that are not shortcuts, and it provides no inherent stopping criteria, requiring that heuristics be used to decide when shortcut edges have been removed. By contrast, we show that graphs with shortcut connections necessarily contain large cycles, and CycleCut guarantees to remove a minimal set of edges to remove all large cycles in the graph.

5.3 CycleCut

The CycleCut algorithm is inspired by max-flow/min-cut methods for graph partitioning. These methods identify the minimal cut to partition a graph by simulating flow across the graph in order to identify the minimal set of edges that limit the flow as illustrated in Figure 5.1. The max-flow min-cut theorem ensures that a cut which removes only the limiting edges is optimal with respect to minimizing the sum capacity of the cut edges [Ford and Fulkerson, 1962].

Instead of simulating flow across a graph, CycleCut simulates flow around a topological hole in order to find a minimal cut that breaks the cycles that enclose the hole, as illustrated in Figure 5.2.

There are three common cases when topological holes occur in sampled manifolds: shortcut edges, toroidal intrinsic variables, and unsampled regions. These cases are illustrated

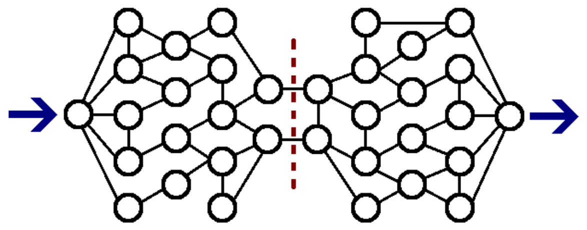


Figure 5.1: Max-flow/min-cut methods simulate flow across a graph to identify a minimal set of edges that limit the flow.

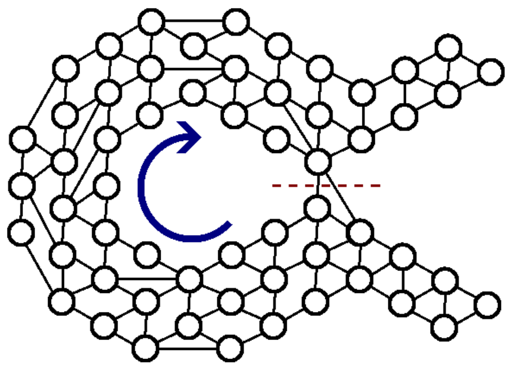


Figure 5.2: CycleCut simulates flow around a topological hole to identify a minimal set of edges that limit the flow.

in Figure 5.3. We will describe these cases using an example of a robot which uses a camera to explore a park. The images that this robot obtains with its camera may be considered to be samples on a manifold. If these samples are "well-behaved", manifold learning can be used to obtain a sequence of intrinsic variables that correspond with the sequence of images. These intrinsic variables will include a representation of the robot's estimated position within the park, which would be useful information for many tasks.

Case 1: Suppose there are two benches in this park with similar appearance. Images of these benches would constitute samples on the manifold with a large geodesic distance, but a small observed distance. Thus, the neighborhood graph of these samples will contain shortcut connections between these observations. These shortcut connections create a topological hole in the sampled manifold. CycleCut will remove the minimal set of edges, such that the topological hole is eliminated. This enables the manifold to be learned without trying to preserve superfluous distances between points.

Case 2: Suppose that the robot has the ability to rotate yaw-wise. Because rotation inherently repeats, the manifold sampled by the robot's images will connect to itself to form an open cylinder topology. Such manifolds cannot be projected into their tangent-space dimensionality without either introducing distortions or breaking continuity. If the cylinder is tall, then using distortion is a particularly poor solution. CycleCut finds a minimal break in continuity to enable the manifold to be unfolded with little distortion. When CycleCut is not used, resulting points are crowded together near the inner circumference and stretched apart near the outer circumference. Distortion-free estimates of a state space are particularly important for planning purposes because the effectiveness of generalization is limited by the consistency within the representation of state.

Case 3: Suppose there is a large pond in the park into which the robot wisely does not advance. This creates an unsampled region which appears to be a topological hole in the manifold, even though the true manifold is continuous. In this case, it would be better not to use CycleCut prior to manifold learning because it is desirable to preserve the unsampled hole as an artifact in the results. If CycleCut is used in this case, it will cut along the shortest path that connects the unsampled region with the outside area. This cut is superfluous, but it will leave a larger amount of edges intact to define the manifold structure. The impact that such a superfluous cut will have on results depends on how much "load" or "tension" is represented in the cut edges. Fortunately, unlike case 2, case 3 typically involves edges that carry little such tension, so results are generally degraded very little. Thus, if case 1 or case 2 is known to also exist in a sampled manifold, or if the nature of a collection of samples is unknown, we recommend using CycleCut prior to manifold learning because the potential cost is low, while the potential benefit is high.

If there are no large sample holes, CycleCut has no effect on results.

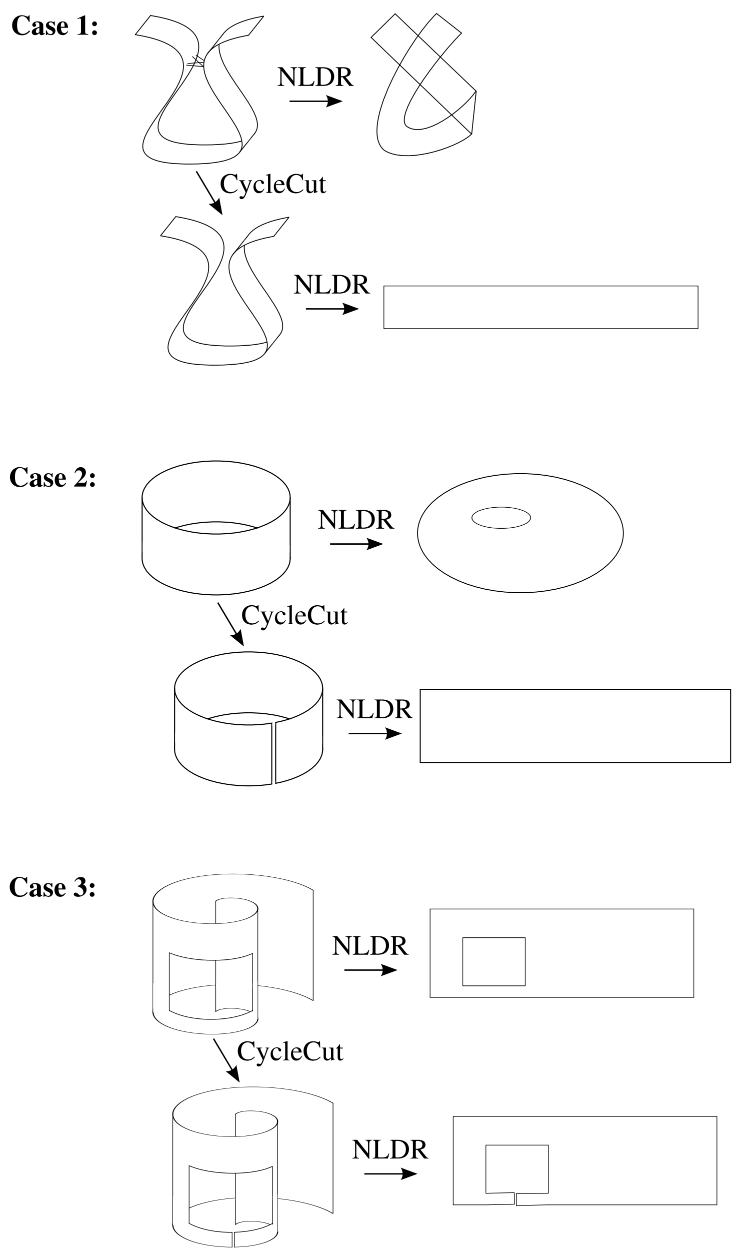


Figure 5.3: *Case 1:* For manifolds with a topological hole due to shortcut edges, CycleCut reduces distortion in the results. *Case 2:* For manifolds with topological holes due to toroidal intrinsic variables, CycleCut reduces distortion at the cost of continuity in the results. *Case 3:* For manifolds with topological holes due to unsampled regions, CycleCut has little effect on results.

5.3.1 The CycleCut algorithm

In graph theory, a chord is an edge that connects two non-adjacent vertices in a cycle. Cycles with no chords are called induced cycles, chordless cycles, or holes. We generalize the notion

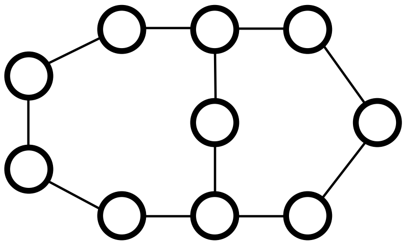


Figure 5.4: This graph contains 3 cycles. All 3 of them are chordless, but only two of them are atomic.

```
function CycleCut(V, E)                                  Comments
    $\lambda \leftarrow 12$                              Default cycle length threshold
1.  for each $\{a,b\} \in E$ do: $W_{a,b} \leftarrow 1$  Initialize edge capacities
    $R \leftarrow$ empty list                            R stores removed edges
    loop:                                                For each large atomic cycles
2.      $C \leftarrow$ find_large_atomic_cycle($V, E, \lambda$)   See Figure 5.6
        if $C =$ null:                                   If no large atomic cycles
            break                                        Exit the loop. Go to *
3.      $h \leftarrow \min_{\{a,b\} \in C} W_{a,b}$      Find the bottle-neck in the cycle
4.      for each $\{a,b\} \in C$:                        For each edge in the cycle
            $W_{a,b} \leftarrow W_{a,b} - h$             Reduce the remaining capacity
5.          if $W_{a,b} = 0$:                            If the edge is fully saturated
                remove $\{a,b\}$ from $E$                Cut the edge
                append $\{a,b\} \rightarrow R$           Remember the removed edges
6.      continue                                         Go to the start of the loop
7.  for each $\{a,b\} \in R$:                            * Repair unnecessary cuts
        add $\{a,b\} \rightarrow E$                      Tentatively restore the edge
        $C \leftarrow$ find_large_atomic_cycle($V, E, \lambda$)   See Figure 5.6
        if $C \neq$ null:                                If the edge creates a cycle
            remove $\{a,b\}$ from $E$                    Remove it again
```

Figure 5.5: Pseudo-code for the CycleCut algorithm. V is a set of vertices. E is a set of edges. (For convenience, an implementation of CycleCut is included in the *Waffles* [Gashler, 2011] machine learning toolkit, and a switch to invoke it has been integrated with each manifold learning algorithm in that toolkit.)

of a chord by defining an n-chord to be a path of length n which connects two vertices in a cycle, where n is less than the length of the shortest path on the cycle between the vertices. We refer to a cycle with no n-chords as an atomic cycle. For example, the graph shown in Figure 5.4 contains 3 chordless cycles, but only two of them are atomic cycles.

The problem of finding large holes in a graph (as defined by a large chordless cycle) has been well-studied [Spinrad, 1991, Nikolopoulos and Palios, 2004]. For the application of preprocessing high-dimensional data for manifold learning, however, large atomic cycles are better indicators of topological holes than large chordless cycles. In high-dimensional space, irregularly distributed samples frequently create sub-structures like the one shown in Figure 5.4. If only 1-chords are considered, then large cycles with n-chords ($n > 1$) will be falsely interpreted to indicate that there is a topological hole in the graph, resulting in unnecessary cuts.

Pseudo-code for the CycleCut algorithm is given in Figure 5.5. This algorithm can be briefly summarized as follows:

1. Equip each edge with a capacity value.
2. Find a large atomic cycle, C, else go to Step 7.
3. Find the smallest capacity, h, of any edge in C.
4. Reduce the capacity of every edge in C by h.
5. Remove all edges with a capacity of zero.
6. Go to Step 2.
7. Restore all removed edges that do not create a large atomic cycle when restored.

The first step of CycleCut equips each edge with a "capacity" value that indicates the *a priori* confidence that it represents a good neighbor connection. In this paper, uniform capacities are used, but non-uniform capacities may be used if additional information is available. The second step finds a large atomic cycle in the graph. (We note that this step could be modified, such that any other undesirable structure would be identified and removed from the graph, but in this paper, we are only interested in removing large atomic cycles.) Section 5.3.2 gives a detailed description of how large atomic cycles are identified. Step 3 finds the minimum capacity, h, of any edge in the large atomic cycle. Step 4 simulates flow by reducing the remaining capacity of every edge in the large atomic cycle by h. Step 5 removes all edges that have become fully saturated. These edges are stored in a list, R. Steps 2 through 5 are repeated until there are no more large atomic cycles in the graph. Finally, Step 7 restores any edges that were unnecessarily removed in step 5. Each edge in R is tentatively restored to the graph. If restoring the edge creates a large atomic cycle, then it is removed again. The resulting cut is guaranteed to be minimal with respect to the sum of capacity values, such that all atomic cycles with a cycle length $\geq \lambda$ are broken.

```
function find_large_atomic_cycle($V, E, \lambda$)          Comments
$Q \leftarrow$ empty queue; $S \leftarrow \emptyset$; $T \leftarrow \emptyset$        $S, T$ = visited vertices, edges
choose a random vertex, $v_0$ from $V$                 Pick a random seed point
enqueue $v_0 \rightarrow Q$; add $v_0 \rightarrow S$                    Seed the outer BFS queue
while $|Q| > 0$:                                       Do the outer breadth-first-search
.   dequeue $a \leftarrow Q$                             Visit the next vertex
.   for each neighbor $b$ of $a$:                        Follow every edge
.       if $b \in S$:                                  If a cycle is detected
.       .   $P_b \leftarrow$ null                              $P$ = parent vertices
.       .   $I \leftarrow$ empty queue; $U \leftarrow$ empty set          $U$ = vertices visited by inner BFS
.       .   enqueue $b \rightarrow I$; add $b \rightarrow U$                Seed the inner BFS queue
.       .   while $|I| > 0$:                           Do the inner breadth-first-search
.       .       dequeue $c \leftarrow I$                     Visit the next vertex
.       .       for each neighbor $d$ of $c$:            Follow every edge
.       .           if $d \notin U$ and $E_{c,d} \in T$:           Limit the inner BFS
.       .               $P_d \leftarrow c$                       Store the parent of every vertex
.       .               enqueue $d \rightarrow I$; add $d \rightarrow U$    Advance the inner BFS
.       .               if $d = a$:                    If an atomic cycle is found
.       .                   $Y \leftarrow$ empty list            $Y$ = vertices in the atomic cycle
.       .                   while $d \neq$ null:           Build the list of vertices
.       .                   .   append $d \rightarrow Y$         Add to the list
.       .                   .   $d \leftarrow P_d$               Advance to parent vertex
.       .                   if $|Y| \geq \lambda$ then return $Y$   Return the large atomic cycle
.       .                   else break twice           Exit inner BFS. Go to *
.       else                                           If b has not been visited before
.       .   enqueue $b \rightarrow Q$; add $b \rightarrow S$                Advance the outer BFS
.       add $E_{a,b} \rightarrow T$                            * Flag this edge as visited
return null                                            There are no large atomic cycles
```

Figure 5.6: Pseudo-code to find an atomic cycle with a cycle length $\geq \lambda$ edges in a graph comprised of vertices V, and undirected edges E.

5.3.2 The *find_large_atomic_cycle* routine

Large atomic cycles are found by iterating over all the atomic cycles in the graph, and returning when one with a cyclelength $\geq \lambda$ is found. Pseudo-code for this routine is given in Figure 5.6. The atomic cycles in a graph are enumerated using a breadth-first-search (BFS) nested within another BFS. We refer to these as "the outer BFS", and "the inner BFS". The purpose of the outer BFS is to define a region in which the inner BFS is constrained during its search. Whenever the outer BFS detects a cycle (by finding an edge that connects to a vertex that it has already discovered), the inner BFS begins searching from that point. The inner BFS is only allowed to follow edges that the outer BFS has already followed (not including the edge that detected the cycle). The inner BFS terminates when it finds a cycle because subsequent cycles that it would discover would either be the same size, or non-atomic.

Proof of correctness

We now give a proof that this routine will find an atomic cycle with a cycle length $\geq \lambda$, if some atomic cycle C, $|C| \geq \lambda$, is reachable from the seed point. Let e be the last edge in C to be traversed by the outer BFS, and let T be the set of edges that had been traversed by the outer BFS just before e was traversed. When the outer BFS traverses e, the destination vertex of e is discovered for at least the second time. This event triggers the inner BFS to find a shortest path, P, from one vertex of e to the other, following only edges in T. Let $B \equiv P \cup \{e\}$. B must not have any n-chords, or else the inner BFS would have arrived at the vertex earlier by cutting across the n-chord. If $|B| = |C|$, then B satisfies the proof, whether or not $B \equiv C$, and the routine returns. To complete the proof, we show that "$|B| \neq |C|$" is an impossible condition. If $|B| > |C|$, then the inner BFS would have found C first, so $B \equiv C$, which is a contradiction. If $|B| < |C|$, then let $A \equiv (B \cup C) \backslash (B \cap C)$, where "$\backslash$" denotes exclusion. A must have been discovered previously, or else e would not close both B and C. Further, A must be an atomic cycle, or else either B or C would have already been closed. If $|A| \geq |C|$, then $|A| \geq \lambda$, so the routine would have returned when it found A. If $|A| < |C|$, then $A \cap B$ is an n-chord of C, which contradicts the assumption that C is atomic.

5.4 Analysis and validation

In this section, we analyze the CycleCut algorithm. In Section 5.4.1, we demonstrate that it is easy to find a good value for λ. In Section 5.4.2, we give a complexity analysis of CycleCut. In Section 5.4.3, we demonstrate the use of CycleCut to remove shortcut connections. In Section 5.4.4, we demonstrate the use of CycleCut to unfold a manifold with a torroidal intrinsic variable.

5.4.1 The threshold

Shortcut edges will always create a large atomic cycle because, by definition, they connect geodesically distant regions of the manifold. As demonstrated in Figure 5.3 Case 3, it is typically not a problem if a few good connections are cut, but it can cause significant problems if bad connections are not cut. Thus, a good value for λ need only be smaller than the cycles caused by shortcut edges. To show that CycleCut is very robust to the value of λ, we sampled a square region of 2D space with 10^6 uniformly distributed random points, and connected each point with its 10-nearest neighbors. (Many manifolds can be viewed as an embedding of this or a similar structure in higher-dimensional space.) We measured the distribution of the cycle lengths of the atomic cycles in this structure. Figure 5.7.A shows the cumulative distribution, with atomic cycle lengths along the horizontal axis and the portion of atomic cycles along the vertical axis. We repeated this experiment with a variety of conditions. In Figure 5.7.B, 5 neighbors were used instead of 10. In Figure 5.7.C, 15 neighbors were used. In Figure 5.7.D, the region was sampled with 100 times fewer points. In Figure 5.7.E, the sampled region was much wider than tall. In Figure 5.7.F, samples were drawn from a Gaussian distribution, instead of a uniform distribution. In Figure 5.7.G, samples were drawn in 3 intrinsic dimensions. In Figure 5.7.H, samples were drawn in 4 intrinsic dimensions. In

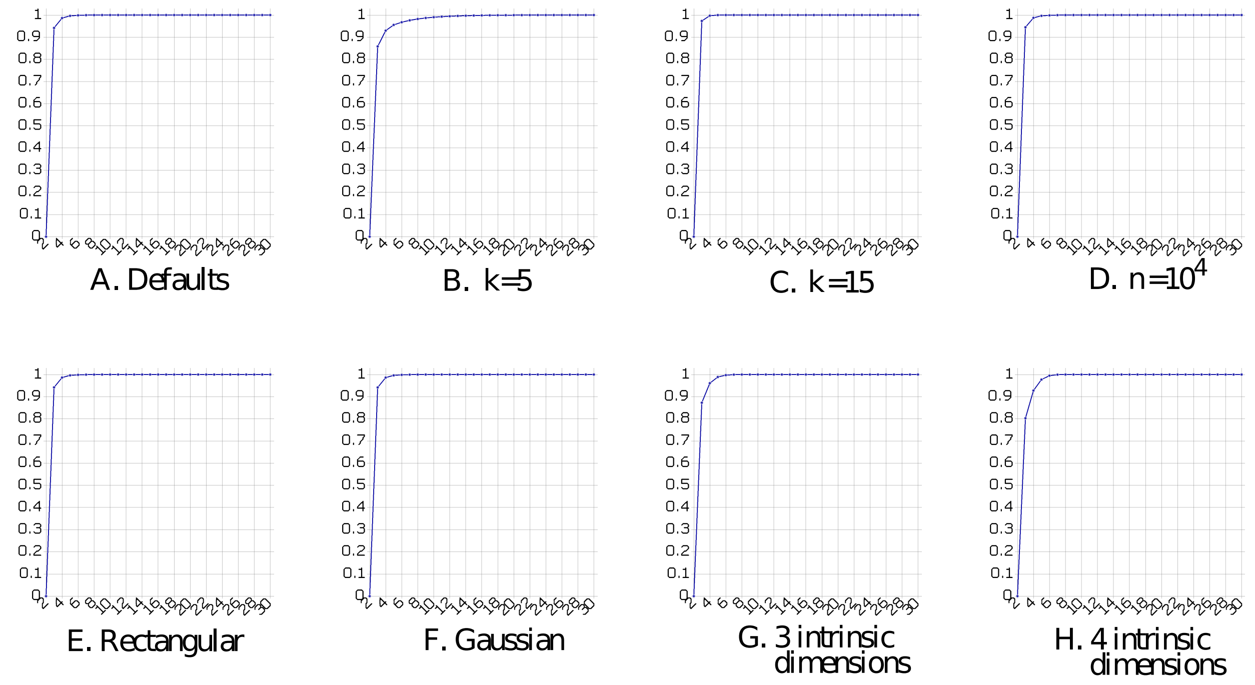


Figure 5.7: The cumulative distribution of the cycle lengths of the atomic cycles from various random distributions of points. In every case, the threshold $\lambda = 12$ includes nearly all of the atomic cycles.

every case, the cumulative distribution was approximately the same. This shows that the threshold value λ is robust across a large variety of conditions. As can be seen in each of the charts in Figure 5.7, the value $\lambda = 12$ is sufficiently large to handle almost all cases. Thus, we used this value in all of our experiments, and it consistently produced good results. In most cases, smaller values, such as $\lambda = 8$, produced identical results as those obtained using $\lambda = 12$.

5.4.2 Complexity

The *find_large_atomic_cycle* subroutine contains an inner BFS nested within an outer BFS. The outer BFS will visit every edge in the graph. The inner BFS will terminate before following a constant number, k^{λ}, of edges, except for the last time it is called, which will terminate before following $|E|$ edges, where E is the set of edges. Thus, the asymptotic computational complexity of *find_large_atomic_cycle* is $O(|E|)$. The number of times that this subroutine must be called depends on the number of shortcut connections in the graph, so the complexity of CycleCut is bounded between $O(|E|)$ and $O(|E|^2)$. If there are few or no shortcuts, CycleCut scales linearly, so it is well-suited to be used as a "safety-net" prior to manifold learning. If there are no shortcut connections, as is typically assumed, then little cost is incurred for performing the check, and if there are some shorcut connections, CycleCut can guarantee to catch them.

5.4.3 Removing shortcuts

We sampled the manifold $\{\sin(2\alpha) + \frac{\alpha}{2}, -2\cos(\alpha), \beta\}$ with 1000 random values for α and β, where α ~Uniform($-\pi$, π), and β ~Uniform(0, 2). Each point was connected with its 14-nearest Euclidean-distance neighbors. 63 of these connections shortcut to geodesically distant regions of the manifold, as shown in Figure 5.8-left. These shortcut connections would significantly affect the results of most manifold learning algorithms in a negative manner. CycleCut correctly detected and removed all 63 shortcut connections. (See Figure 5.8-right.) No other connections were cut. (This is an example of case 1 from Figure 5.3.)

In another experiment, we sampled a manifold by uniformly drawing 50×50 pixel sub-images from a larger image of rocky terrain. (See Figure 5.9.A.) Because a certain region on one side of this image was made visually similar to a certain region on the other side, this manifold approaches very close to itself. Figure 5.9.B shows results obtained using Isomap with 12 neighbors to reduce 4680 sample sub-images into 2 dimensions. These values are a poor estimate of the window position. In many cases, it ambiguously assigns similar window positions for very different sub-images. Much better results were obtained using the same algorithm and parameters when the neighborhood graph was preprocessed with CycleCut. (See Figure 5.9.C.) CycleCut removed the connection between some of the images, even though they were very similar, because they created an atomic cycle in the graph. The results with CycleCut are a better estimate of the window position, and are sufficiently unambiguous

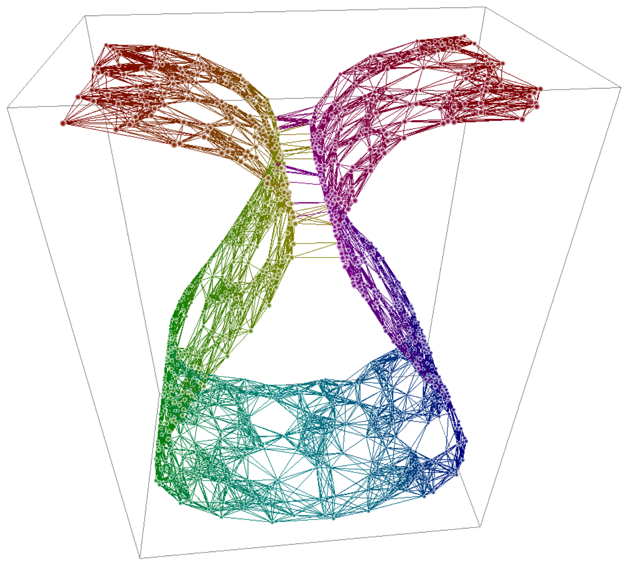


Figure 5.8: *Left:* 1000 points are shown connected with each of their 14-nearest neighbors. 63 of the connections shortcut to geodesically distant regions of the manifold. *Right:* CycleCut removed all of these shortcut connections without removing any other connections.

that a model could easily be trained using the resulting data to map from window position to the sub-image, or from the sub-image to the window position.

5.4.4 Toroidal manifolds

We sampled a panoramic image of a courtyard using a 40×30 pixel sliding window with 3 color channels at 225×31 unique window positions. (See Figure 5.10.) We then used Isomap with 12 neighbors to reduce this data from 3600 to 2 dimensions, both with and without CycleCut. (This is an example of case 2 from Figure 5.3.) In the case where CycleCut was not used, Isomap returned results that ambiguously represented multiple window positions in several regions of the reduced space. When CycleCut was used, it returned results that approximately corresponded to the yaw and pitch represented by the sliding window.

We repeated this experiment using a different panoramic base image. In this case, we used an image of Barrage de Malpasset, courtesy of Wikimedia Commons. (See Figure 5.11.A.) As in the previous experiment, each sub-image corresponds with the image that a camera would view from this location if pointed in a particular direction. We used Isomap to

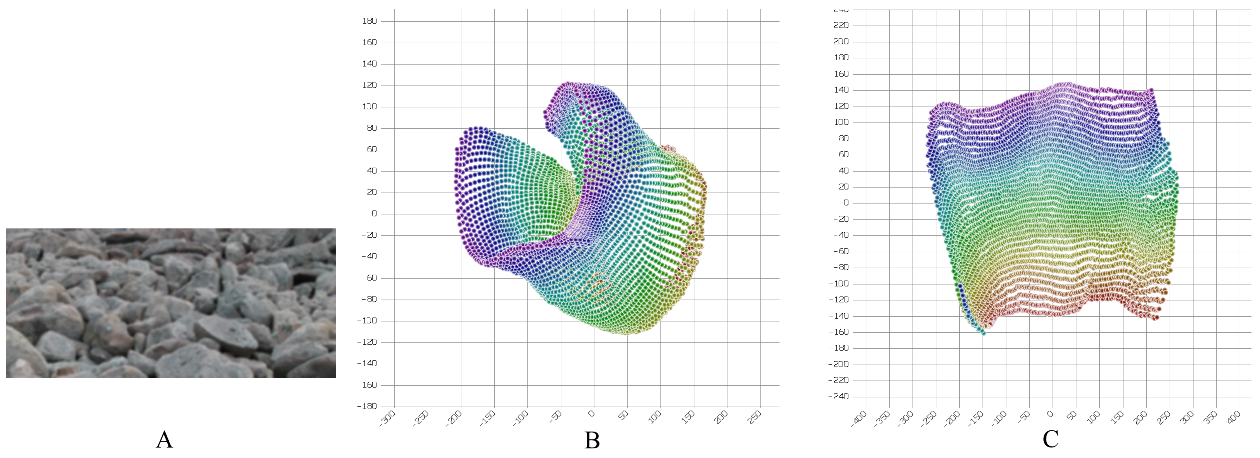


Figure 5.9: *A:* An image of rocky terrain. We uniformly took 4680 samples from the manifold of 50 × 50 pixel sub-images with 3 color channels of this terrain. Because a certain region on one side of this image is visually similar to a certain region on the other side, this manifold approaches very close to itself. *B:* Results obtained using the Isomap NLDR algorithm with 12 neighbors to reduce these 4680 samples into 2 dimensions. These values are a poor estimate of the window position. In many cases, it ambiguously predicts similar window positions for very different sub-images. *C:* Results obtained using the same algorithm and the same number of neighbors, but also using CycleCut to preprocess the neighborhood graph. These results are a better estimate of the window position, and are sufficiently unambiguous that a model could easily be trained using this data to map from window position to the image, or from the image to the window position.

estimate the window position (or camera direction) by reducing the collection of sub-images to two dimensions. These results are plotted in Figure 5.11.B. In this case, Isomap made its projection by flattening most of the predicted positions, but stretching a certain region of them to connect to both ends. Unfortunately, this results in positions having non-unique meaning. When CycleCut was used, however, it minimally severed the manifold such that it could be unfolded without ambiguity. (See Figure 5.11.C.) The “ripples” that appear in these results are due to the assumption Isomap makes that distances in observation space are proportional to distances in state space. (CycleCut does not correct any assumptions made by NLDR algorithms. It only corrects topological issues in the manifold itself.)

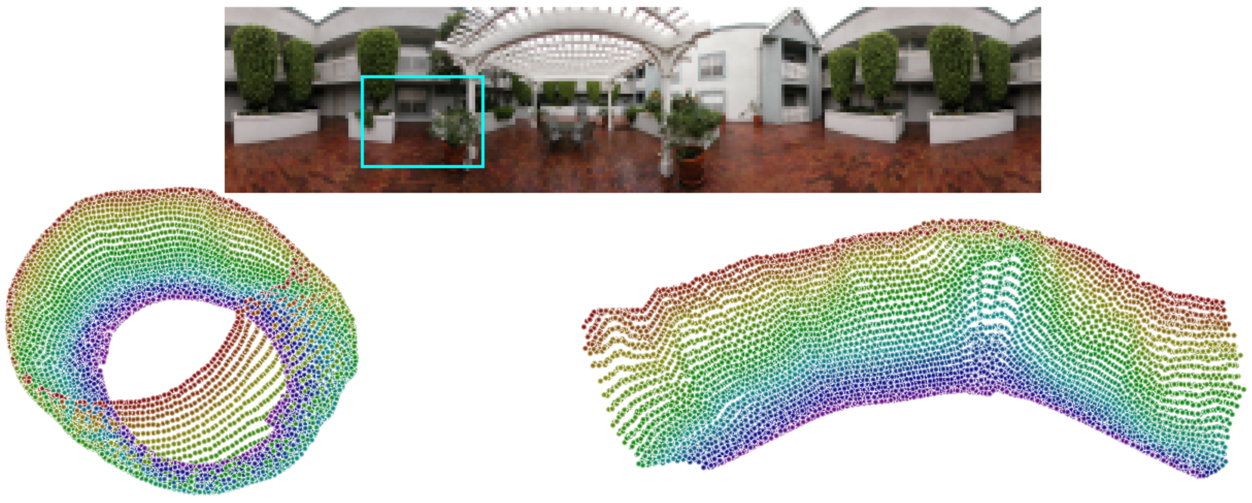


Figure 5.10: *Top:* A 40 × 30 pixel window was sampled with 3 color channels at 225 × 31 window positions within a panoramic image of a courtyard. (Image derived from a photo by Rich Niewiroski Jr. from the Wikimedia Commons.) *Left:* A projection of this data by Isomap [Tenenbaum et al., 2000] with 12 neighbors into 2 dimensions. Each point is colored according to the vertical position of the window. *Right:* A projection by the same algorithm after CycleCut was applied to the neighborhood map. These values are more meaningful with respect to the yaw and pitch represented by the window.

When not to use CycleCut

In cases where intrinsic variables are torroidal, CycleCut reduces distortion at the cost of preserving continuity. Consequently, the choice of whether to use CycleCut when torroidal intrinsic variables exist depends on which is more important. In visualization tasks, for example, it may be desirable to present results both with and without CycleCut. If the two visualizations differ, then the one produced with CycleCut is likely to represent local structure more faithfully, while the one without CycleCut is likely to present a better global view of the continuity within the data. In tasks where it is desirable to separate intrinsic variables into meaningful attributes, such as the state estimation tasks that we demonstrate in this paper, CycleCut tends to lead to results that exhibit a more linear structure. In such cases, the original neighorhood graph can be used to provide the continuity information that CycleCut rejects. Finally, as we demonstrate in the next section, if it is known that all large

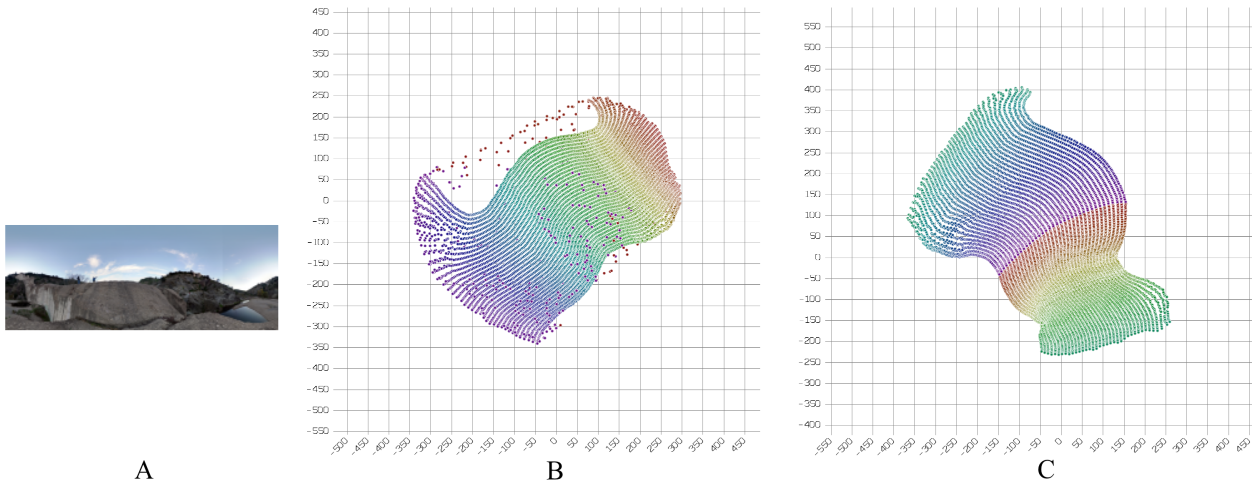


Figure 5.11: *A:* A panoramic image of Barrage de Malpasset, courtesy of Wikimedia Commons. We uniformly sampled a manifold by drawing sub-images from this panoramic image. Each sub-image corresponds with the image that a camera would view from this location if pointed in a particular direction. *B:* We used Isomap to estimate the window position (or camera direction). In this case, Isomap made its projection by flattening most of the predicted positions, but stretching a certain region of them to connect to both ends. Unfortunately, this results in positions having non-unique meaning. *C:* CycleCut minimally severed the manifold such that it could be unfolded without ambiguity. (The "ripples" that appear in these results are due to an assumption Isomap makes that distances in observation space are proportional to distances in state space. CycleCut does not correct faulty assumptions made by NLDR algorithms.)

atomic cycles in the graph are due to unsampled regions, then there is little value in using CycleCut.

5.4.5 Unsampled Regions

We sampled a Swiss Roll manifold, $\{(8\alpha + 2)\sin(8\alpha), (8\alpha + 2)\cos(8\alpha), 12\beta\}$, with 2000 random points. A star-shaped region of this manifold was excluded from being sampled. We then used Manifold Sculpting [Gashler et al., 2008b] to reduce the dimensionality of these points. Figure 5.12.A shows the original points. Figure 5.12.B shows the results obtained from reducing the dimensionality to 2, without using CycleCut. Figure 5.12.C shows results when CycleCut was used to preprocess neighborhood connections. The results obtained with CycleCut were nearly identical to those obtained without it. This shows that CycleCut often has little adverse effect in cases where it is not needed. (This experiment is an example of case 3 from Figure 5.3.)

In another experiment, we simulated a robot navigating within a warehouse environment by sampling in a windowed region from an image of a warehouse. (See Figure 5.13.A.) We simulated forward and reverse movement by changing the size of the window, and we simulated lateral movement by moving the window horizontally. Linear interpolation was used to produce a uniform-sized view from the perspective of the simulated robot. We applied 4000 random actions from the set {move forward (scale down the window size), move back (scale up the window size), move left, move right} to this robot and collected the corresponding images from its view. We simulated a physical obstacle by preventing this robot from entering a rectangular region near the middle of its state space. We also injected Gaussian noise into both the transitions and the observations to simulate unreliable hardware. Specifically, we added a random value from a Gaussian distribution with a standard deviation of 5 percent of the channel range to each color channel of each pixel in each observed sub-image. We also added Gaussian noise with a standard deviation of 5 percent of the step length to each state transition, such that the simulated robot would transition to a

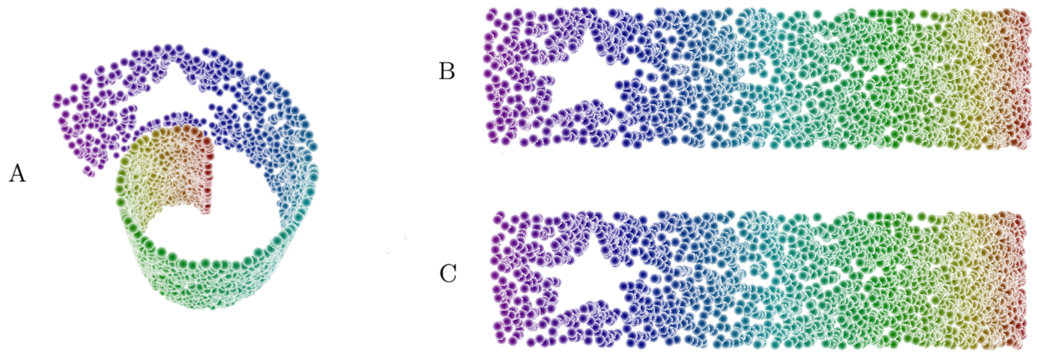


Figure 5.12: *A:* 2000 sample points on the surface of a Swiss Roll manifold with a star-shaped region excluded from sampling. *B:* Results obtained by reducing this dataset to two dimensions with *Manifold Sculpting* [Gashler et al., 2008b]. *C:* Results obtained with Manifold Sculpting after CycleCut was applied. In this case, 28 neighbor connections were cut, but the results are nearly identical to those obtained without CycleCut. This shows that CycleCut has little adverse effect in cases where it is not needed.

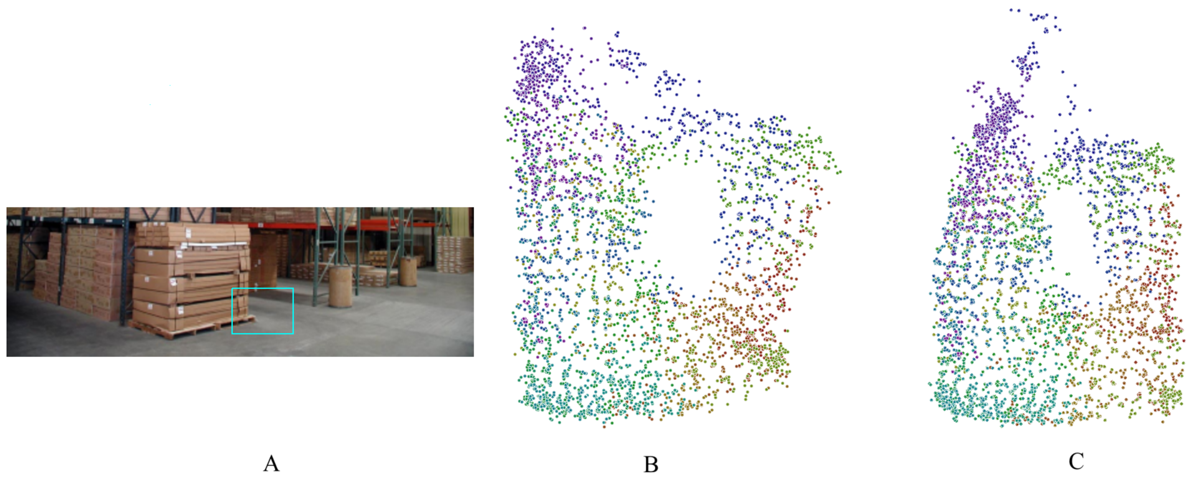


Figure 5.13: *A:* We simulated a robot navigating within a warehouse environment by sampling this image within a window. We simulated forward and reverse movement by changing the size of the window, and we simulated lateral movement by moving the window horizontally. Linear interpolation was used to produce a uniform-sized view from the perspective of the simulated robot. We applied 4000 random actions from the set {move forward, move back, move left, move right} to this robot and collected the corresponding images. We simulated a physical obstacle by preventing this robot from entering a rectangular region near the middle of its state space. We also injected Gaussian noise into both the transitions and the observations. *B:* We used Temporal NLDR to reduce these images to estimates of the robot's position. *C:* We repeated this procedure with the addition of CycleCut. In this case, the ideal results would be unchanged. Although CycleCut did change the results somewhat, the overall structure of the state estimates is very similar whether or not CycleCut was used. The most significant difference occurs near the top region, where only a small number of samples were taken (because the robot only wandered into that region one time).

state only near to the one specified by the actions that it performed. After collecting 4000 observation images, we used the Temporal NLDR algorithm [Gashler and Martinez, 2011b] to reduce them to corresponding estimates of the robot's position. These results are plotted in Figure 5.13.B. The "hole" in the center of these results indicates that the robot did not collect any observations in that region of its state space, as intended. We then repeated this experiment with the addition of CycleCut to pre-process the neighborhood graph. The results with CycleCut exhibited similar overall structure, as shown in Figure 5.13.C. Some differences, however, appear in the upper region of this result. This occured because that portion of the state space was poorly sampled, because the robot only wandered into that region one time, so the manifold learning algorithm had little structural information to preserve in that region.

5.5 Conclusions

We presented an algorithm called CycleCut, which finds a minimal cut necessary to break the large atomic cycles in a graph. We demonstrated, both theoretically and empirically, that this algorithm is useful for preparing neighborhood graphs for manifold learning. With many typical problems, neighborhood graphs contain no large atomic cycles. In such cases, CycleCut has no effect on the results. When neighborhood graphs do exhibit large atomic cycles, however, dimensionality reduction can be difficult. In these cases, CycleCut finds a minimal set of connections that must be cut to eliminate all large atomic cycles from the graph.

CycleCut makes existing non-linear dimensionality reduction algorithms robust to a wider range of problems. Using CycleCut to preprocess neighborhood graphs incurs little cost because it scales well, it has little adverse effect on results in the case where large atomic cycles are expected in the neighborhood graphs, and it guarantees never to break connectivity. When shortcut connections exist, CycleCut guarantees to remove them, and when the manifold connects back to itself, CycleCut has the beneficial effect of removing the minimal number of connections such that the manifold can be unfolded without introducing local distortions.

Chapter 6

Tangent Space Guided Intelligent Neighbor Finding

Abstract: We present an intelligent neighbor-finding algorithm called SAFFRON that chooses neighboring points while avoiding making connections between points on geodesically distant regions of a manifold. SAFFRON identifies the suitability of points to be neighbors by using a relaxation technique that alternately estimates the tangent space at each point, and measures how well the estimated tangent spaces align with each other. This technique enables SAFFRON to form high-quality local neighborhoods, even on manifolds that pass very close to themselves. SAFFRON is even able to find neighborhoods that correctly follow the manifold topology of certain self-intersecting manifolds.

6.1 Introduction

Many algorithms commonly used in machine learning rely on local neighborhoods of points. Instance-based learners (IBL), for example, use consensus among neighboring points to determine a predicted label for previously unseen points. As another example, non-linear dimensionality reduction (NLDR) algorithms measure the distances between neighboring points to represent structure on the surface of a manifold. The quality of results obtained by these algorithms is limited by the quality of the local neighborhoods with which they operate. When data is known to lie on the surface of a low-dimensional manifold in high-dimensional space, poor neighborhood connections are typically defined as those that shortcut across the manifold structure between geodesically distant regions of the manifold [Balasubramanian and Schwartz, 2002, Varini et al., 2006].

We present a novel algorithm called Similarly Aligned Friend Finding RelaxatiON (*SAFFRON*) that is designed to find neighbors on the surface of a non-linear manifold. SAFFRON takes advantage of the assumption that the samples are drawn from the surface of a lower-dimensional manifold to select neighbors in a manner that intelligently avoids short-cutting to geodesically distant regions of the manifold surface. This is useful because many interesting data sets, including collections of images or documents, tend to form a non-linear manifold, and SAFFRON enables good neighbors to be found in such data, even when the manifold structure passes very near to, or in some cases even intersects with, itself. SAFFRON makes IBL and NLDR algorithms suitable for use with a wider class of problems–specifically, those that sample highly folded manifolds. In cases where the manifold structure does not fold back on itself, SAFFRON gives results similar to those obtained with Euclidean distance. Thus, it is well-suited for general-purpose use.

As a simple example of data that might lie on a manifold that passes very near to itself, we consider a hypothetical collection of images that is obtained by a robot with a camera as it travels down a hallway inside a building. It might be useful to use an NLDR algorithm to reduce this collection of images into fewer dimensions because the intrinsic variables in this data would correspond with the robot's position and orientation. Unfortunately, in many buildings, hallways contain doors that are very similar in appearance to each other. Thus, if Euclidean distance is used to find neighboring images, it might incorrectly determine that two images are very similar, even though they depict very distant positions with the hallway. These images may be considered to be samples from a highly-folded manifold with a topology that passes very near to itself. Existing techniques may not be able to find local neighborhoods on such a topology that correctly represent the structure of the manifold.

The SAFFRON algorithm seeks to intelligently select local neighborhoods that correctly represent the manifold structure, particularly when the manifold passes very close to itself or

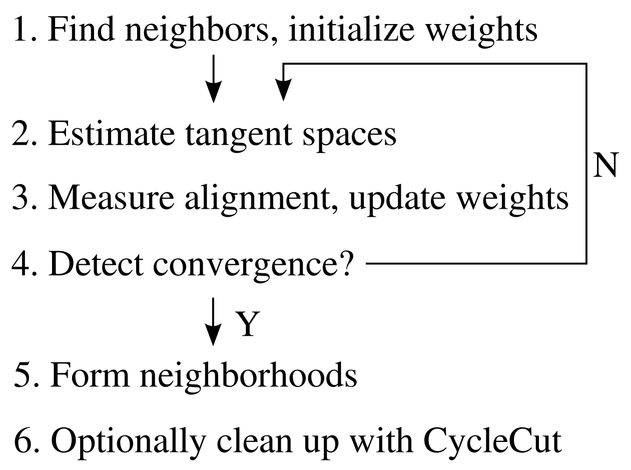

1. Find neighbors, initialize weights
2. Estimate tangent spaces
3. Measure alignment, update weights
4. Detect convergence?
N
Y
5. Form neighborhoods
6. Optionally clean up with CycleCut

Figure 6.1: A high-level flow diagram of the SAFFRON algorithm.

even intersects itself. We assume that a set of points, $\mathbf{P}$, has been sampled on the surface of a manifold, such that each point $\mathbf{p}_i \in \mathbf{P}$ is a vector in $\Re^d$. We also assume that the tangent-space dimensionality, t, of the manifold is known. (In cases where t is not known, methods exist for estimating this value from the data [Levina and Bickel, 2005].) We refer to the points that SAFFRON determines to be compatible as *friends*, rather than *neighbors*, because they are determined by their *compatability*, rather than just by their proximity. SAFFRON determines two points to be compatible if the tangent spaces associated with them are closely aligned.

In order to accurately measure the tangent space at a point, the structure of the manifold would need to be known *a priori*. Unfortunately, most existing techniques for learning a manifold structure rely on first finding neighborhoods to represent the local structure. Thus, in order to identify points that are well-suited to be labeled as friends, SAFFRON uses a relaxation technique that alternately estimates the tangent space for each point from its current set of friends, and then refines the friends based on the alignment of their tangent spaces. When convergence is detected, SAFFRON can intelligently identify local neighborhoods in a manner that is unlikely to shorcut across the manifold topology. A high-level flow diagram of the SAFFRON algorithm is given in Figure 6.1.

The SAFFRON algorithm is comprised of six high-level steps. We now briefly discuss each of these high-level steps. A more detailed specification of the algorithm, including all details necessary for implementation, is given in Section 6.3. The first high-level step of the SAFFRON algorithm uses Euclidean distance to find a set of neighbors, or candidate friends, $\mathbf{C}_i$. $|\mathbf{C}_i|$ should be larger than the number of friends, k, that SAFFRON seeks. A weight vector, $\mathbf{w}_i$ specifies the affinity between $\mathbf{p}_i$ and each of its candidate friends in $\mathbf{C}_i$. Initially, each candidate point is given uniform weight. As the system relaxes in subsequent steps, the weight will tend to shift toward the k points in $\mathbf{C}_i$ whose tangent spaces are most aligned with that of $\mathbf{p}_i$.

The second high-level step of SAFFRON estimates the tangent space at each $\mathbf{p}_i \in \mathbf{P}$. This is done by computing the t-first principal components of $\mathbf{C}_i$. The weight of each neighbor is used when computing the principal components, such that as a candidate point gains more weight, it will have more influence on the estimate of the tangent space.

The third step is the most significant part of SAFFRON, so we describe it in greater detail. This step measures how well the tangent-space of $\mathbf{p}_i$ is aligned with the tangent-space of each candidate point, and uses this information to update the weights. In order to establish a suitable metric for determining how well the tangent spaces of two points are aligned, we define two types of angles as illustrated in Figure 6.2.

The monohedral (one-surface) angle is defined between a point, $\mathbf{p}_i$, with its corresponding tangent space, $\mathbf{S}_i$, and another point, $\mathbf{p}_j$. $\mathbf{S}_i$ is represented such that each row specifies one of the orthonormal basis vectors in the tangent space of $\mathbf{p}_i$. The monohedral angle is computed by projecting point $\mathbf{p}_j$ onto $\mathbf{S}_i$, and then computing the angle formed by the three points $\langle \mathbf{p}_j, \mathbf{p}_i, \mathbf{p}_i + \mathbf{S}_i(\mathbf{p}_j - \mathbf{p}_i) \rangle$. Equation 6.1 computes the cosine of the monohedral angle.

$$m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j) = \frac{||\mathbf{S}_i(\mathbf{p}_j - \mathbf{p}_i)||}{||\mathbf{p}_j - \mathbf{p}_i||} \tag{6.1}$$

The dihedral (two-surfaces) angle is defined between two tangent spaces, $\mathbf{S}_i$ and $\mathbf{S}_j$. It is independent of the points $\mathbf{p}_i$ and $\mathbf{p}_j$. In the case where $t = d - 1$, the dihedral angle

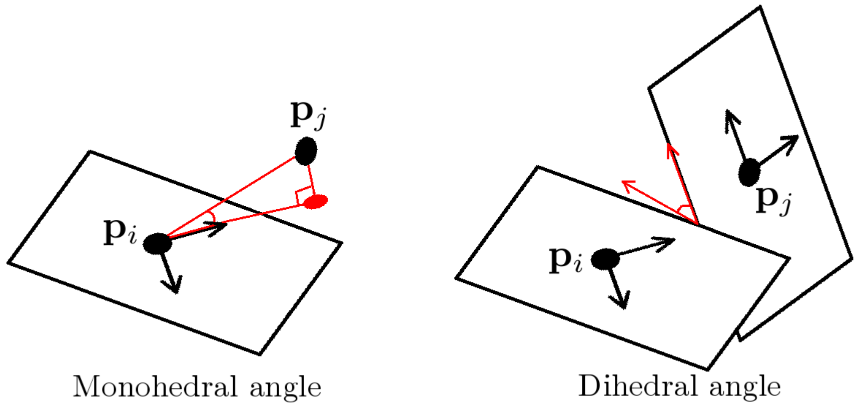


Figure 6.2: The monohedral (one surface) angle is defined between a point with its corresponding tangent space, and another point. It is independent of the second point's tangent space, and is not commutative. The dihedral (two surface) angle is defined between two tangent spaces. It is independent of the points, and is commutative.

is simply the angle formed by the normal vectors of the two surfaces. Typically, however $t < d - 1$. In this case, we use the normal vectors that maximize their distance with their projection onto the other surface. This can be found with an eigenvector optimization technique. Equation 6.2 computes the cosine of the dihedral angle between two tangent spaces. This equation works even when the tangent spaces have a codimensionality greater than 1. The function $\mathrm{fpc}(\mathbf{M})$ as used in Equation 6.2 returns the first principal component of the row vectors in $\mathbf{M}$, or the eigenvector with the largest eigenvalue of the covariance matrix of $\mathbf{M}$.

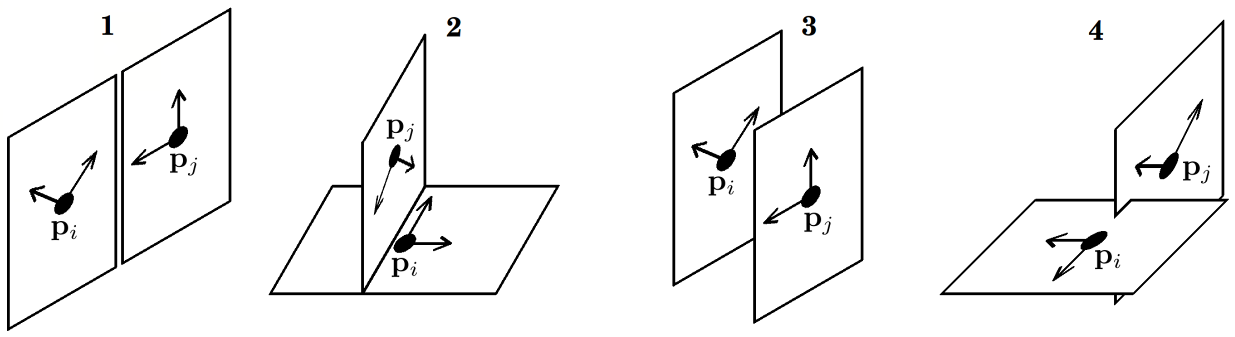
$$\begin{aligned} d(\mathbf{S}_i, \mathbf{S}_j) = {} & \left(\mathbf{S}_i \,\mathrm{fpc}(\mathbf{S}_j^T \mathbf{S}_j \mathbf{S}_i^T - \mathbf{S}_i^T)\right) \cdot \\ & \left(\mathbf{S}_j \,\mathrm{fpc}(\mathbf{S}_i^T \mathbf{S}_i \mathbf{S}_j^T - \mathbf{S}_j^T)\right) \end{aligned} \tag{6.2}$$

SAFFRON uses Equation 3 to measure how well two tangent spaces are aligned. When $\mathbf{S}_i$ and $\mathbf{S}_j$ are mis-aligned, Equation 6.3 will return a value close to 0. When they are aligned, it will return a larger value. The monohedral angle is measured in both directions because it is not commutative, while the dihedral angle need only be evaluated once because it is commutative. The dihedral component of this equation will be small when the two tangent spaces are not approximately parallel. At least one of the two monohedral components will be small when the tangent spaces are approximately parallel, but have a large gap between them. Thus, when the product of all three components is large, the tangent spaces are necessarily aligned, and so Equation 6.3 provides a useful indication of whether the Euclidean distance between two points is representative of the geodesic distance between them, given the estimated tangent spaces, or whether those points actually lie on separate regions of a manifold that happens to fold back on itself to create a misleading proximity between the two points.

$$\begin{aligned} a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j, \mathbf{S}_j, \lambda) = {} & \min(\lambda, m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j)^2) * \\ & \min(\lambda, m(\mathbf{p}_j, \mathbf{S}_j, \mathbf{p}_i)^2) * \\ & \min(\lambda, d(\mathbf{S}_i, \mathbf{S}_j)^2) \end{aligned} \tag{6.3}$$

Equation 6.3 uses the squared cosine of these angles instead of just the cosine of these angles because both positive and negative correlations are equally indicative of alignment. In order to be tolerant of noise and curvature in the manifold, we cap the value of all three components with a threshold, λ, such that all nearly-aligned tangent spaces are evaluated as being equally good. λ can range from 0 to 1, but for most problems, suitable values typically range from about 0.85 to 0.95. Smaller values will cause the metric to be more tolerant of curvature in the manifold, while larger values will cause the metric to be more careful to avoid connecting two points from geodesically distant regions of the manifold. As λ approaches 0, the results from SAFFRON approach those obtained using Euclidean distance to find neighbors.

Figure 6.3 gives an intuition for why the product of these three components provides a good measure of tangent-space alignment. This figure shows 4 cases. In each case, the two



	Case 1	Case 2	Case 3	Case 4
$m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j)$	≈ 1	≈ 0	≈ 0	≈ 1
$m(\mathbf{p}_j, \mathbf{S}_j, \mathbf{p}_i)$	≈ 1	≈ 1	≈ 0	≈ 1
$d(\mathbf{S}_i, \mathbf{S}_j)$	≈ 1	≈ 0	≈ 1	≈ 0
$a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j, \mathbf{S}_j, \lambda)$	$\approx \lambda^3$	≈ 0	≈ 0	≈ 0

Figure 6.3: Four cases are shown with two points separated by a constant distance and their corresponding tangent spaces. Case 1 is the only case where the tangent spaces of the two points are aligned. The cosine of the monohedral angle is evaluated in both directions because it is not commutative, whereas the cosine of the dihedral angle is the same in both directions. The tangent spaces of two points are aligned when all three metrics are close to 1. This property generalizes into arbitrary dimensional spaces.

points are nearly the same distance apart, but their corresponding tangent spaces are oriented in various ways. For the purpose of visualization, we enclose each point with a rectangular region on its corresponding tangent space. Arrows extending from the point represent a set of basis vectors that might define this tangent space. In case 1, where the tangent spaces are aligned, all three values are close to 1. In each case where the the tangent spaces are misaligned in some way (cases 2, 3, and 4), at least one of the three values is close to zero. Thus, the product of these three components is only close to 1 when the tangent spaces are aligned.

SAFFRON decays the weights of the $|\mathbf{C}_i| - k$ candidate friends that are the least aligned with $\mathbf{p}_i$. The weight vector is then normalized to sum to 1. (This effectively shifts the weight toward the k candidate friends with tangent spaces that are most aligned with $\mathbf{p}_i$.)

Step 4 of the SAFFRON algorithm detects convergence. Convergence can be easily detected because each iteration increases the alignment scores of the candidate friends with the largest weights. Thus, a goodness score for the system is given as the sum of the alignment scores scaled by the weights. When this goodness value no longer increase significantly, it has converged.

Step 5 forms local neighborhoods of points. This is done for each point by selecting the k candidate friends with the largest weights.

The last high-level step of SAFFRON is to post-process the neighborhoods with the *CycleCut* algorithm [Gashler and Martinez, 2012]. CycleCut is an algorithm that uses a max-flow/min-cut technique to detect and prune neighbor connections that shortcut across the manifold topology. We consider a complete description of the CycleCut algorithm to be outside the scope of this paper, but we give a high-level overview of its function here. The CycleCut algorithm is described in Chapter 5.

This step may be considered to be optional because with many problems, SAFFRON can form high-quality neighborhoods without this last step. The addition of this step, however, increases the robustness of the SAFFRON algorithm. SAFFRON relies on several parameter values (including the number of candidate points q, the number of neighbors k, the number of tangent-space dimensions t, and a threshold value, λ). When these values are poorly-tuned for the problem, or when the data contains a large amount of noise, SAFFRON may find a small number of neighbors that still shortcut across the manifold structure. CycleCut can detect and remove these connections, but it can only distinguish the shortcuts from valid connections if the number of shortcut connections is small. Thus, SAFFRON and CycleCut are designed to complement each other. SAFFRON makes it possible for CycleCut to identify shortcut connections by keeping the number of shortcut connections small, and CycleCut ensures that the final results will be free of the large cycles that are created by connections that shortcut across manifold boundaries.

The remainder of this paper is layed out as follows. Section 6.2 describes related work. Section 6.3 describes the SAFFRON and CycleCut algorithms in detail sufficient to facilitate implementation. Section 6.4 reports results from empirical tests to validate the properties of SAFFRON. Finally, Section 6.5 summarizes the contributions of this paper.

6.2 Related Work

Due to the large number of algorithms that rely on being able to identify neighboring points, neighbor-finding has been a topic of interest in machine learning for a long time. Many distance, similarity, and dis-similarity metrics have been proposed for this purpose [Mahalanobis, 1936, Menger, 1942, Stigler, 1989, Diday, 1974, Wilson and Martinez, 1997]. It has been shown that as dimensionality becomes large, the distance from a point to its farthest neighbor approaches the distance to its nearest neighbor [Jonathan et al., 1999]. Thus, techniques that intelligently select neighbors, rather than merely relying on a distance metric, are particularly important for applications involving high-dimensional space. In this paper, we focus on finding neighbors among points that are known to lie on the surface of an intrinsically low-dimensional manifold in high-dimensional space.

The idea of using the assumption that data lies on a manifold to guide the evaluation of distances is not new [Nomizu and Ozeki, 1961]. Recently, however, as interest in manifold learning has increased in machine learning communities, techniques for estimating distances between points that sample from an unknown manifold have become of interest. One example is *Ranking on Manifolds* [Zhou et al., 2004]. This technique evaluates distance in a manner that penalizes paths that cross gaps over unsampled regions of the manifold, and thereby evaluates distance in a manner that conforms more closely to the intrinsic manifold structure. Perhaps the most similar algorithm to SAFFRON is Adaptive Neighborhood Selection for Manifold Learning [Wei et al., 2008]. This technique builds on top of Ranking on Manifolds by adaptively choosing parameter values in local neighborhoods. SAFFRON is similar in that it also adaptively chooses settings in each neighborhood, but differs in how it determines proximity to the manifold. SAFFRON uses an approach similar to that used by the Local Tangent Space Alignment (LTSA) [Zhang and Zha, 2002] for estimating a tangent space by computing the first t principal components in local neighborhoods. Unlike LTSA, however, it does so without the assumption that neighbors have already been found, and it uses the tangent spaces to guide the selection of neighbors (or friends). This approach enables SAFFRON to find better local neighborhoods on the surface of sampled manifolds than existing techniques.

6.3 The SAFFRON algorithm

Figure 6.4 gives pseudocode for the SAFFRON algorithm. The parameters to the algorithm are:

$\mathbf{P} \equiv$ set of points on which to operate.

$q \equiv$ median number of candidate points from which to select friends.

$k \equiv$ number of friends to find for each point. ($k < q$.)

$t \equiv$ dimensionality of the manifold tangent space.

$\lambda \equiv$ threshold that determines tolerance to curvature.

Line 1 of the SAFFRON algorithm computes a radius, r, in which to find candidate friends for each point. Because a suitable radius is necessarily problem-specific, it is preferable to parameterize the algorithm in terms of q, the median number of candidates, and to compute r from q. **Lines 2-5** find a set of candidate friends for each point and assign uniform weight to each of them. These candidates consist of all points within the specified radius of each point. **Lines 6, 10, and 20** compute a "goodness" score for the system that is used in line 21 to dectect convergence. The value of g will increase with each iteration until the system converges. **Lines 7-21** are the main loop of the SAFFRON algorithm. **Lines 8-9** estimate the tangent space of each point by computing the first t principal components of $\mathbf{C}_i$. Each candidate neighbors is weighted according to $\mathbf{w}_i$ when the principal components are computed. Each row in $\mathbf{S}_i$, therefore, will store one of the t orthonormal basis vectors in the tangent space for $\mathbf{p}_i$. **Lines 11-19** update the weights of each point in $\mathbf{P}$. **Line 15** uses Equation 6.3 to compute an alignment score for each candidate friend of $\mathbf{p}_i$. **Line 18** decays the weight of the candidate friends with the lowest alignment scores. Our implementation decays these weights by multiplying by 0.9. Obviously other values could be used here, but we have not been able to detect any significant effect on results by adjusting this value. If there are multiple candidates with the same alignment score, the closer points are considered to be better aligned. **Line 19** normalizes $\mathbf{w}_i$ to sum to one. Thus, lines 18 and 19 together effectively shift the weight toward the k-best friends of $\mathbf{p}_i$. **Line 21** detects convergence. Our implementation stops when the goodness score increases by less than 0.01% after one iteration over all of the data points. Other values could be used to detect convergence, but we have obtained little perceptible benefit by tuning this value. **Line 22** forms the local neighborhoods by selecting the k points with the largest weights to be the friends of each $\mathbf{p}_i$. **Line 23** performs a post-processing operation on the neighborhoods called *CycleCut*, which detects and removes any shortcut connections that were erroneously found in previous steps. CycleCut is only effective at detecting shortcut connections if the number of shortcut connections is small, so it is typically not sufficient to just use CycleCut by itself. The CycleCut algorithm is described in Chapter 5.

6.4 Experimental Analysis

To show that SAFFRON is effective at forming neighborhoods that do not shortcut across the manifold, even when the manifold approaches very close to itself, we sampled 439 points uniformly on an intrinsically one-dimensional compressed helix manifold defined by the equations $\langle \sin(\alpha), 0.005\alpha, \cos(\alpha) \rangle$. Figure 6.5.A shows each point connected with its four nearest Euclidean-distance neighbors. If the manifold is considered to be an intrinsically two-dimensional ring, then the neighbors found by Euclidean distance correctly represent its topology. Unfortunately, Euclidean distance does not provide a mechanism to specify

```
     function $\mathbf{SAFFRON}(\mathbf{P}, q, k, t, \lambda)$
1    set $r \leftarrow$ the median distance to the $q^{th}$ neighbor in $\mathbf{P}$.
2    for each point $\mathbf{p}_i \in \mathbf{P}$:
3        let $\mathbf{C}_i$ be the subset of points $\mathbf{p}_j \in \mathbf{P}$, where
             $j \neq i$, and $||\mathbf{p}_j - \mathbf{p}_i|| < r$.
4        let $\mathbf{w}_i$ be a vector of weights for points in $\mathbf{C}_i$.
5        set each $w_{ij} \in \mathbf{w}_i \leftarrow \frac{1}{|\mathbf{w}_i|}$.
6    $g \leftarrow 0$
7    loop:
8        for each point $\mathbf{p}_i \in \mathbf{P}$:
9            $\mathbf{S}_i \leftarrow$estimate_tangent_space$(\mathbf{P}, i, \mathbf{C}_i, \mathbf{w}_i, t)$.
10       set $h \leftarrow g$; $g \leftarrow 0$.
11       for each point $\mathbf{p}_i \in \mathbf{P}$:
12           let $\mathbf{x}_i$ be a vector of alignment scores
                 for each candidate point in $\mathbf{C}_i$.
13           for each point $\mathbf{c}_{ij} \in \mathbf{C}_i$:
14               let $\mathbf{p}_l$ be the point in $\mathbf{P}$ that is $\mathbf{c}_{ij}$.
15               set $x_{ij} \leftarrow a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_l, \mathbf{S}_l, \lambda)$
16           set $e \leftarrow |\mathbf{x}| - k$.
17           for each of the $e$-smallest values $x_{ij} \in \mathbf{x}_i$:
18               set $w_{ij} \leftarrow 0.9 * w_{ij}$.
19           set $\mathbf{w}_i \leftarrow \frac{\mathbf{w}_i}{\sum_j w_{ij}}$.
20           set $g \leftarrow g + \mathbf{x}_i \cdot \mathbf{w}_i$.
21   until $\frac{g}{h} - 1 < 0.0001$
22   for each $\mathbf{p}_i \in \mathbf{P}$, choose the $k$ candidate points with the
         largest weights to be the friends of $\mathbf{p}_i$.
23   Prune any shorcut connections with the
     CycleCut algorithm.
```

Figure 6.4: Pseudo code for finding the k-best friends of each row in $\mathbf{P}$, assuming the points lie on a t-dimensional manifold. r is a radius in which to find candidate points. λ is a threshold value between 0 and 1 that specifies how well-aligned the tangent-spaces of points must be in order for them to be considered friends.

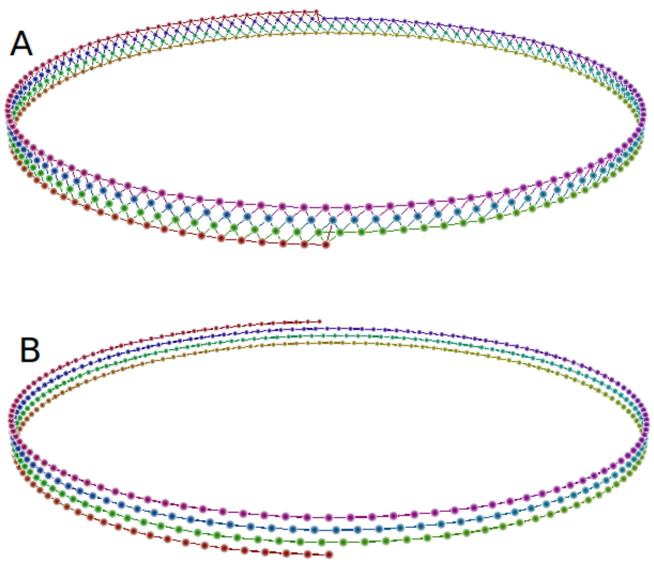


Figure 6.5: **A:** Samples at regular intervals are shown on an intrinsically one-dimensional compressed helix manifold. Each point is shown connected with its four nearest Euclidean-distance neighbors. These would represent good local neighborhoods for an intrinsically two-dimensional ring manifold, but Euclidean distance does not provide a mechanism to specify the intrinsic dimensionality. **B:** Each point is shown connected with its four best friends as determined by SAFFRON with parameters $q = 32$, $k = 4$, $t = 1$, $\lambda = 0.95$.

the intrinsic dimensionality. With SAFFRON, this is done by setting the parameter value t. Figure 6.5.B shows the same points connected with their 4-best friends, as determined by SAFFRON with parameters $q = 32$, $k = 4$, $t = 1$, $\lambda = 0.95$.

We note that a solution to a compressed helix manifold using adaptive neighborhood selection was also presented by [Wei et al., 2008]. In that case, however, the compressed helix was compressed only to the point where the nearest neighbors of each point still included the previous and next points in the one-dimensional sequence. The compressed helix manifold for which we report results is a much harder problem because it is sufficiently compressed that neither the previous nor next point in the sequence is even among the four nearest Euclidean-distance neighbors for many of the points. By aligning tangent spaces, however, SAFFRON is able to correctly determine which points are most likely to be neighboring samples on a manifold with the specified number of intrinsic dimensions. If t is set to 2 instead of 1, SAFFRON behaves more like Euclidean distance by choosing neighbors in vertical directions as well as horizontal directions.

The parameters that we used to solve this problem were selected intuitively. The value $q = 32$ was chosen such that the pool of candidate friends for each point would be sufficiently large to include the two previous and two next points in the helix. The value $k = 4$ was chosen for the visual appeal of displaying 4 neighbors in Figure 6.5.A. The value $t = 1$ was chosen to indicate the intended intrinsic dimensionality of the problem. The value $\lambda = 0.95$ was chosen to be fairly large in order to prevent neighbors from shortcutting across the manifold. With these parameters, SAFFRON made no shortcutting neighbor connections. The same results were obtained both with and without the final CycleCut shortcut-pruning step of the SAFFRON algorithm. When the CycleCut step is included, however, a broader range of parameter values can be used to still obtain the same results.

To test SAFFRON with an even more extreme manifold, we sampled 63 points from an intrinsically one-dimensional manifold that intersects with itself according to the equations $\langle \sin(2\alpha), 2\cos(\alpha) \rangle$. Figure 6.6.A shows each point connected with its 4-nearest Euclidean-

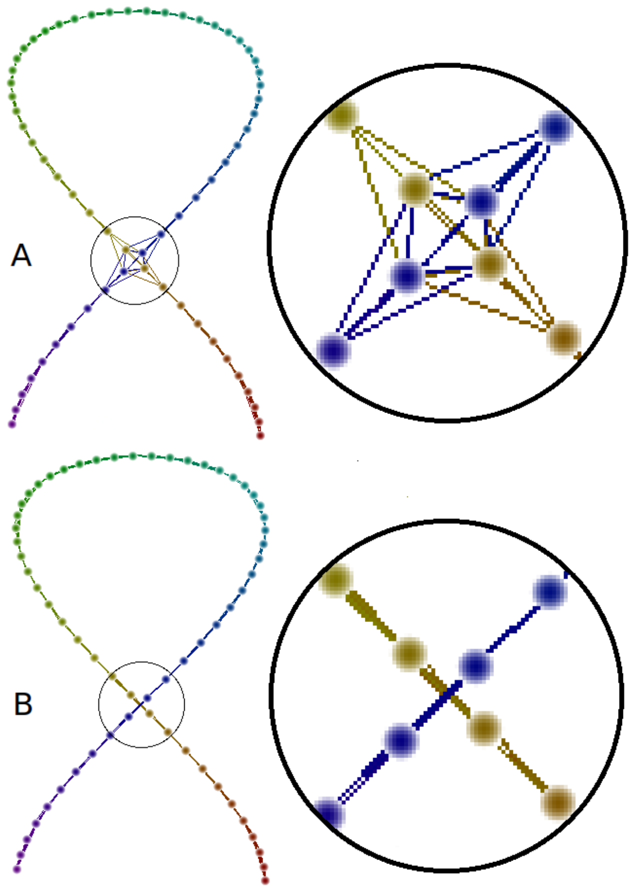


Figure 6.6: **A:** Samples at regular intervals are shown on an intrinsically one-dimensional self-intersecting manifold. Each point is shown connected with its four nearest Euclidean-distance neighbors. These connections shortcut across the manifold structure, misrepresenting the manifold topology. NLDR algorithms that rely on such neighborhoods will not correctly unfold the manifold. **B:** Each point is shown connected with its four best friends as determined by SAFFRON with parameters $q = 6$, $k = 4$, $t = 1$, $\lambda = 0.9$.

distance neighbors. These connections shortcut across the manifold structure, misrepresenting the manifold topology. If an NLDR algorithm were to try to unfold this manifold based on these neighborhoods, the shortcut connections would cause it to produce incorrect results. By contrast, Figure 6.6.B shows the same points connected with their 4-best friends as determined by SAFFRON with parameters $q = 6$, $k = 4$, $t = 1$, $\lambda = 0.9$. None of the neighbor connections found by SAFFRON shortcut across the manifold topology. SAFFRON is able to determine that the very close points near the center would make poor friends because they have very mis-aligned tangent spaces. To our knowledge, SAFFRON is the first neighbor-finding technique that can correctly find neighbors on a self-intersecting manifold. The same results were obtained both with and without the final CycleCut shortcut-pruning step of the SAFFRON algorithm. When the CycleCut step is included, however, a broader range of parameter values can be used to still obtain the same results.

We also created an intrinsically two-dimensional manifold embedded in three-dimensional space according to the equations $\langle \sin(2\alpha), 2\cos(\alpha), 2\beta \rangle$. With this manifold, we sampled 2000 points using random values for α and β from a uniform distribution. Figure 6.7.A shows a plot of these point. We used SAFFRON to compute local neighborhoods with the parameters $q = 40$, $k = 18$, $t = 2$, $\lambda = 0.9$. We then used Manifold Sculpting [Gashler et al., 2008b] to reduce the dimensionality of these points to two dimensions, using the local neighborhoods computed by SAFFRON. Figure 6.7.B shows a plot of the data after reducing to two dimensions. In this case, the CycleCut step was necessary to obtain this result.

For visual clarity, the points in Figure 6.7 are colored according to the value of α. SAFFRON, of course, did not have access to any information that would enable it to determine the color of any of the points, or the corresponding intrinsic values α or β. It can be observed in Figure 6.7.B that a small number of purple-colored points were incorrectly placed among the yellow points, and a small number of yellow points were incorrectly placed among the purple point. This occurs because these points were located very near to where the manifold intersected itself. As the relaxation process used by SAFFRON proceeds, the

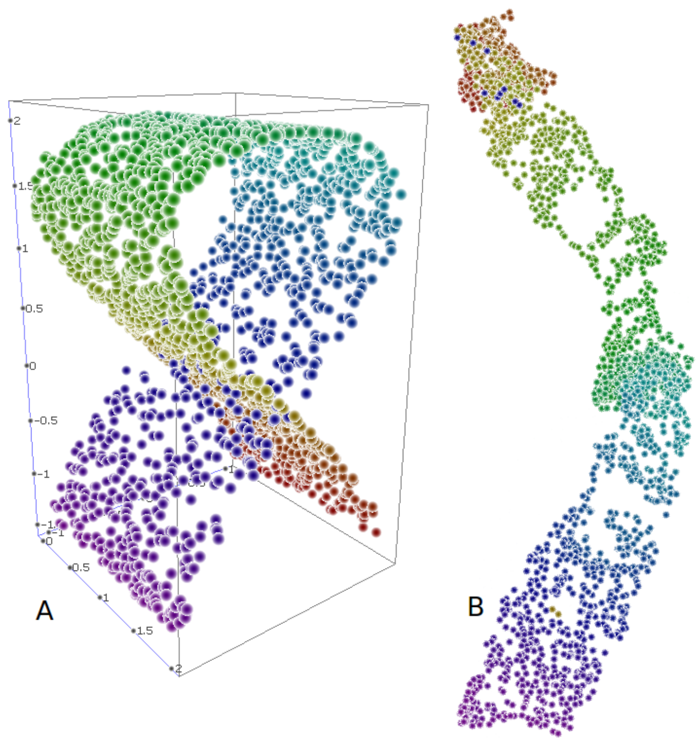


Figure 6.7: **A:** 2000 sample points on an intrinsically two-dimensional self-intersecting manifold. **B:** SAFFRON was used with the parameters $q = 40$, $k = 18$, $t = 2$, $\lambda = 0.9$ to find neighbors, and then Manifold Sculpting [Gashler et al., 2008b] was used to reduce the dimensionality to 2. Because these neighborhoods did not jump across manifold boundaries, Manifold Sculpting was able to correctly unfold this manifold.

tangent spaces at each point align themselves with their neighbors. With these points, the correct alignment could not be unambiguously determined. As these points began to align with one side, they also simultaneously disassociated themselves with the other side, and therefore became entirely embedded into just one location of the graph. A less-intelligent neighbor-finding technique might leave these points connected to both regions of the manifold. This would create a graph that an NLDR algorithm would not be able to unfold because the shortcut connections would bind geodesically distant regions of the manifold close together.

6.5 Conclusion

We presented an intelligent neighbor-finding algorithm called SAFFRON, which uses a relaxation technique to select neighbors such that the tangent spaces represented in local neighborhoods will be aligned. This technique enables neighbors to be found that are robust against shortcutting across manifold structure. Because so many important techniques used in machine learning rely on graphs formed by connecting neighboring points, SAFFRON has significant potential to improve the analysis of data that lies on a low-dimensional manifold in high-dimensional space.

The contributions of this paper include: a method for measuring the dihedral angle between two flats with a codimension greater than 1, a method for evaluating the alignment of two tangent spaces by computing the product of the dihedral and monohedral angles, and most significantly, a relaxation technique for determining which points are best-suited to be neighbors, without shortcutting across a manifold structure.

Part III

Applied Manifold Learning Techniques

The chapters in this part present new manifold learning techniques designed for specific applications, and implementations thereof. Following the rush of enthusiasm for manifold learning that was manifested in machine learning communities around the turn of the century (2000), a certain sense of pessimistic counter-response seemed to follow as researchers began to note that few practical applications were emerging for manifold learning. Although some expectations may have initially been set too high, another part of this counter-response may have been caused by a shortage of patience, as we demonstrate in this part that manifold learning is a critical component in solutions to certain significant applications.

Chapter 7 demonstrates a new manifold learning technique designed for modeling dynamical systems. In order to make manifold learning suitable for this task, it challenges one of the key assumptions made by nearly all existing manifold learners, that distances in observation space are proportional to distances in state space. By eliminating this poor assumption, we show that manifold learning can be used to estimate the state of dynamical systems with precision from image-based observations. We also show that this leads to a method for training recurrent neural networks, which outperforms existing methods by a significant margin.

Chapter 8 presents a manifold learning approach that is particularly well-suited for the task of imputing missing values in data. By assuming that data points lie on the surface of a manifold, this technique is able to produce a better intrinsic representation of each pattern, and this naturally leads to better predictions of missing values.

Chapter 9 reports on an open source toolkit of machine learning algorithms, called Waffles. This toolkit implements all of the algorithms described in this dissertation, as well as several existing dimensionality reduction algorithms. It also makes several contributions outside of dimensionality reduction, that are not available in other machine learning toolkits.

Chapter 7

Temporal Nonlinear Dimensionality Reduction

Abstract: Existing Nonlinear dimensionality reduction (NLDR) algorithms make the assumption that distances between observations are uniformly scaled. Unfortunately, with many interesting systems, this assumption does not hold. We present a new technique called *Temporal NLDR* (TNLDR), which is specifically designed for analyzing the high-dimensional observations obtained from random-walks with dynamical systems that have external controls. It uses the additional information implicit in ordered sequences of observations to compensate for non-uniform scaling in observation space. We demonstrate that TNLDR computes more accurate estimates of intrinsic state than regular NLDR, and we show that accurate estimates of state can be used to train accurate models of dynamical systems.

7.1 Introduction

Nonlinear dimensionality reduction (NLDR) algorithms operate on an unordered set of high-dimensional observation vectors, $\mathbf{Y} = \langle \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n \rangle$. They compute a corresponding set of low-dimensional vectors, $\mathbf{X} = \langle \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \rangle$, such that each $\mathbf{x}_i$ is a low-dimensional representation of $\mathbf{y}_i$. $\mathbf{x}_i$ may be thought of as a representation of the intrinsic values, or state, from which the corresponding observation $\mathbf{y}_i$ was derived.

In order to compute $\mathbf{X}$, most NLDR algorithms make the assumption that the pair-wise distances between neighboring points in $\mathbf{Y}$ will be approximately proportional to the corresponding pair-wise distances in $\mathbf{X}$. We refer to this as *the assumption of proportional distances*. Unfortunately, in many interesting cases, this assumption does not hold. For example, consider a robot which uses a camera to navigate within a building. Each $\mathbf{y}_t \in \mathbf{Y}$ is a high-dimensional vector of pixel values obtained from the robot's camera. When the robot moves, the amount of change in the observation is not only affected by the amount of movement, but also by the distance between the camera and the objects that are in view. Further, the color and texture patterns of the objects in view may cause observational changes to be scaled higher or lower, independent of the actual change in state. Partial occlusions may further exacerbate this problem. Because the assumption of proportional distances does not hold in many real-world systems, traditional NLDR algorithms are not suitable for estimating the intrinsic state of these systems.

We present a new technique called *Temporal NLDR* (TNLDR), which utilizes the additional information found in sequences of observations to perform nonlinear dimensionality reduction without making the assumption of proportional distances. We show that TNLDR is effective at estimating the intrinsic state of systems from high-dimensional observations, where traditional NLDR algorithms fail. We also demonstrate that an accurate estimate of intrinsic system state is useful for training a model of the system, a process also known as nonlinear black-box system identification.

TNLDR operates using more information than regular NLDR. TNLDR assumes that $\mathbf{Y}$ is an ordered sequence of observations occurring at regular time intervals. Additionally, a sequence of actions, $\mathbf{A} = \langle \mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n \rangle$, is given to specify the control inputs given to influence the dynamical system. By using this additional information, TNLDR is able to compute pair-wise distances between observations in a manner that is independant of the local scaling of observations, and then use existing methods to estimate system state.

TNLDR is designed to reduce the dimensionality of the output of dynamical systems. A general model of a dynamical system is given in Figure 7.1. At any given time, t, the output, $\mathbf{y}_t$, can be observed to provide information about the system. We assume that $\mathbf{y}_t$ is

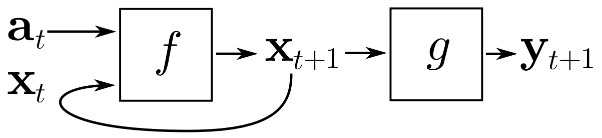


Figure 7.1: A model of a dynamical system. f is the transition function of the system. g is the observation function. t specifies a time-step. $\mathbf{a}$ is an action vector, which may be discrete or continuous. $\mathbf{x}$ is a vector of continuous values that represent state. $\mathbf{y}$ is a high-dimensional observation vector.

a high-dimensional vector of continuous values. $\mathbf{y}_t$ is not directly a function of the current input $\mathbf{a}_t$. Rather, $\mathbf{y}_t$ is a function of the hidden state, $\mathbf{x}_t$, which may be considered to be an aggregation of the initial state, $\mathbf{x}_0$, and all of the inputs, or actions, that were previously applied to the system, $\langle \mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{t-1} \rangle$. Actions may be continuous or discrete. At each time-step, the system transitions to a new state, which is determined by the current state and the action which is applied to the system. Thus, traditional supervised learning algorithms, which assume the output is a direct function of the input, are not sufficient for modeling dynamical systems. Dynamical systems can be challenging to model because the sequence of states, $\mathbf{X} = \langle \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{n-1} \rangle$, is not directly observable. TNLDR, however, can greatly simplify the task of modeling a dynamical system by estimating a sequence of states, $\mathbf{X}$, that corresponds with the sequence of actions, $\mathbf{A}$, applied to the system, and the sequence of observations, $\mathbf{Y}$, that come out from the system. Thus, TNLDR estimates $\mathbf{X}$, from $\{\mathbf{A}, \mathbf{Y}\}$. In the robot example, if $\mathbf{A}$ is a sequence of n actions from the set {turn left, turn right, move forward, back up}, and $\mathbf{Y}$ is the video of n frames from the robot's camera, then TNLDR might use this information to estimate a sequence of vectors that represent the position and orientation of the robot at each of the n time-steps.

In order for TNLDR to compensate for non-uniform local scaling in observation space, it needs a mechanism to estimate how distances are scaled at each position. Therefore, a regression model, h, is trained with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n$. h gives an estimate of how the observation vector will change when a particular action is applied at any position in observation space. The magnitude of this predicted change corresponds with the local scaling of observations. Typically, NLDR algorithms compute the pair-wise distance between two observations, $\mathbf{y}_i$ and $\mathbf{y}_j$, as $||\mathbf{y}_j - \mathbf{y}_i||$. This approach is simple, but it does not compensate for local scaling. TNLDR computes this distance by finding the most direct path of actions from $\mathbf{y}_i$ to $\mathbf{y}_j$, using h to estimate the effect of each candidate action at every step along the path. The distance between $\mathbf{y}_i$ and $\mathbf{y}_j$ is the number of actions in the path, or the number of time-steps that separate the two observations. This time-based distance metric is better-suited for analyzing the high-dimensional observations from dynamical systems because it is independent of the factors that cause observations to be scaled disproportionately with state. TNLDR uses this time-based distance metric to choose local neighborhoods, and to compute the pair-wise distances, $\mathbf{D}$, between them, where each $d_{ij} \in \mathbf{D}$ is the time-based distance between $\mathbf{y}_i$ and $\mathbf{y}_j$. TNLDR then uses a regular NLDR to compute $\mathbf{X}$ from $\mathbf{D}$. TNLDR may be summarized at a high-level as:

1. Train a regression model, h, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$.

2. Use h to identify neighboring observations, and to compute the pair-wise distances, $\mathbf{D}$, between neighboring observations.

3. Use a regular NLDR algorithm to compute $\mathbf{X}$ from $\mathbf{D}$.

TNLDR is designed to analyze high-dimensional observations from dynamical systems. This may be useful, for example, because it will enable a system to be monitored using a general-purpose optical camera, which produces high-dimensional information, instead of problem-specific sensory devices.

We report results from several experiments that demonstrate the effectiveness of TNLDR. We compare TNLDR with existing NLDR algorithms, and show that TNLDR obtains better results. We also show that TNLDR can be used to facilitate the training of models of dynamical systems that have external controls. We compare models trained by TNLDR with those trained using conventional methods and show that the models trained with TNLDR are more accurate. We show that TNLDR enables other regression models, besides neural networks, to be used in a recurrent manner for modeling dynamical systems. We also demonstrate that a model trained with TNLDR is sufficiently accurate to facilitate planning in isolation from the system.

This paper is laid out as follows. In Section 7.2 we give an overview of related work. In Section 7.3 we describe TNLDR in detail. Section 7.4 reports analysis and validation of TNLDR and the related SEIT method. Finally, Section 7.5 summarizes the contributions of this paper.

7.2 Related Work

Many works have been presented in the last decade related to NLDR [Tenenbaum et al., 2000, Roweis and Saul, 2000, Zhang and Zha, 2002, Donoho and Grimes, 2003, Weinberger et al., 2004, Levina and Bickel, 2005, Venna and Kaski, 2006, Gashler et al., 2008b]. In our experiments, we use Isomap [Tenenbaum et al., 2000] and Breadth-first Unfolding with TNLDR, although other NLDR algorithms are suitable as well. The primary focus of TNLDR is for analyzing high-dimensional nonlinear observations from dynamical systems, especially when an estimate of state is required. The notion of using dimensionality reduction to estimate state has been used previously, particularly for the application of robot tracking [Black, 1996, Crowley et al., 1998, Pourraz and Crowley, 1998, Nayar et al., 1996, Keeratipranon et al., 2006]. To our knowledge, however, no dimensionality reduction technique has yet tried to specifically make use of the additional information that is found in sequences of observations.

In order to demonstrate the effectiveness of TNLDR, we use TNLDR to train models of dynamical systems. Most work to date related to modeling nonlinear dynamical systems involves recurrent neural networks. Existing methods for training the weights of recurrent neural networks can be broadly divided into two categories: Those based on nonlinear global optimization techniques, and those based on descending a local error gradient. Perhaps the most common nonlinear global optimization technique for training recurrent neural networks is evolutionary optimization [Floreano and Mondada, 1994, Sjöberg et al., 1995, Blanco et al., 2001]. Unfortunately, in practice, evolutionary optimization tends to be extremely slow, and it is unable to yield good results with many difficult problems [Sontag, 1993, Sjöberg et al., 1995]. Evolutionary optimizers are particularly susceptible to problems where an error surface exhibits narrow channels. The optimizer will typically become stalled as it waits to find a lucky vector that falls within the narrow region of improved vectors. Gradient-based methods that converge to a local optimum offer much faster convergence than optimization techniques that seek the global optimum. Perhaps the most popular of the gradient-based techniques for recurrent networks is Backpropagation Through Time (BPTT) [Mozer, 1995]. Another common gradient-based method is Real-Time Recurrent Learning (RTRL) [Robinson and Fallside, 1987]. Although gradient-based methods converge faster than global optimization methods, they are susceptible to problems with local optima. With feed-forward neural networks, local optima is often considered to be an insignificant problem since many local optima occur near relatively good regions of the weight space. With recurrent neural networks, however, local optima is a much more significant problem [Cuéllar et al., 2006]. The feedback which comes through the recurrent connections can create fluctuating and chaotic responses in the error surface, causing local optima to occur both frequently and in poor locations of the weight space. By contrast, the NLDR component of TNLDR is designed specifically for estimating intrinsic state without being subject to problems with local optima. We show that this naturally leads to better models than can be obtained using BPTT and other methods.

TNLDR may be loosely comparable with the extended Kalman filter (EKF), since both are used to estimate the hidden state of a system. The EKF, however, utilizes a non-linear model of system dynamics, provided by the user, to estimate state. By contrast, TNLDR estimates state without requiring a model of the system to be supplied. Thus, the state estimated by TNLDR may be used to train a non-linear model of system dynamics.

7.3 The TNLDR algorithm

TNLDR is described in pseudo-code in Figure 7.2. We now describe each step in the algorithm.

```
function TNLDR($\mathbf{Y}, \mathbf{A}$)
    Let $n = |\mathbf{Y}| = |\mathbf{A}| =$ the length of time represented in the training data.
    Let $\mathbf{D}$ be a neighbor-distance table, where each $d_{ij} \in \mathbf{D}$ represents the
        distance between neighboring observations $\mathbf{y}_i$ and $\mathbf{y}_j$.
    $\epsilon \leftarrow 2$
1.  Train a regression model, $h$, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$.
2.  for each $\mathbf{y}_i \in \mathbf{Y}$:
        for each $\mathbf{y}_j \in \mathbf{Y}$, $j \neq i$:
            $\mathbf{p}(i, j) \leftarrow$FindPath($\mathbf{y}_i, \mathbf{y}_j$), where FindPath is defined in Figure 7.3
            if $||\mathbf{p}(i, j)|| < \epsilon$ then $d_{ij} = ||\mathbf{p}(i, j)||$ else $\mathbf{y}_j$ is not a neighbor of $\mathbf{y}_i$
3.  Use an NLDR algorithm to compute a sequence of context vectors, $\mathbf{X}$, from $\mathbf{D}$.
```

Figure 7.2: Pseudo-code for TNLDR.

Step 1: The first step of TNLDR is to train a supervised learning regression algorithm, h, with each $\langle \mathbf{y}_t, \mathbf{a}_t \rangle \rightarrow (\mathbf{y}_{t+1} - \mathbf{y}_t)$, where $t < n - 1$. In our implementation, we use a 1-nearest-neighbor model for h, but other algorithms may be suitable as well.

Step 2: A path of actions, $\mathbf{p}(i, j)$, is estimated from each $\mathbf{y}_i$ to $\mathbf{y}_j$, where $j \neq i$, $j < n$, and $i < n$. In our implementation, we use the simple greedy approach described with pseudo-code in Figure 7.3 to find this path, but other path search algorithms may be suitable as well. Our simple greedy path search constructs a set, $\mathcal{B}$, of all the unique actions in $\mathbf{A}$. At each step, it removes all actions from $\mathcal{B}$ that are not positively correlated with the remaining difference in observations. This ensures that it does not find paths that spiral inward or overshoot and then backtrack. It always chooses the action that is most positively correlated with the remaining difference, which is not necessarily the action that advances closest to the goal. This approach is more robust when it is not known how the axes are locally scaled with respect to each other in observation space. It will take at most one step before recomputing the predicted change in observations. Fractional steps are permitted when it is less than one time-step away from the goal.

The path returned by this greedy algorithm is a vector of the number of times that each action is performed on the estimated most-direct path from $\mathbf{y}_i$ to $\mathbf{y}_j$. This is a lossy

```
function FindPath($\mathbf{y}_i, \mathbf{y}_j$)                                    Comments
$r \leftarrow ||\mathbf{y}_j - \mathbf{y}_i||$                                      Measure the initial residual
$\mathcal{B} \leftarrow$ the set of unique actions in $\mathbf{A}$                   Make a set of candidate actions
$\mathbf{p}(i,j) \leftarrow \mathbf{0}^{|\mathcal{B}|}$                              Start with empty path
while $|\mathcal{B}| > 0$ :                                                          While there are useful actions
    for each $\mathbf{b} \in \mathcal{B}$:                                           Prune counter-productive actions
        if $h(\mathbf{y}_i, \mathbf{b}) \cdot (\mathbf{y}_j - \mathbf{y}_i) \leq 0$  If b is not positively correlated
            remove $\mathbf{b}$ from $\mathcal{B}$                                   Do not try action b again
    if $|\mathcal{B}| > 0$ :                                                         If there are still useful actions
        $\mathbf{a} \leftarrow \text{argmax}_{\mathbf{b} \in \mathcal{B}} \frac{h(\mathbf{y}_i,\mathbf{b})}{||h(\mathbf{y}_i,\mathbf{b})||} \cdot (\mathbf{y}_j - \mathbf{y}_i)$   Find the best-correlated action
        $s \leftarrow \min(1, \frac{h(\mathbf{y}_i,\mathbf{a}) \cdot (\mathbf{y}_j - \mathbf{y}_i)}{||\mathbf{y}_j - \mathbf{y}_i||^2})$   Compute fractional action
        $\mathbf{y}_i \leftarrow \mathbf{y}_i + s * h(\mathbf{y}_i, \mathbf{a})$    Step closer to $\mathbf{y}_j$
        $p_{\mathbf{a}}(i,j) \leftarrow p_{\mathbf{a}}(i,j) + s$                     Update the count for action a
if $||\mathbf{y}_j - \mathbf{y}_i|| < \lambda * r$ then return $\mathbf{p}(i,j)$      Accept good estimates
else return null                                                                     Reject very poor estimates
```

Figure 7.3: Pseudo-code for a greedy algorithm that estimates $\mathbf{p}(i,j)$, where $p_{\mathbf{a}}(i,j)$ is the number of times that action a is performed on the estimated most-direct path from $\mathbf{y}_i$ to $\mathbf{y}_j$. (Fractional counts are permitted.) The value $\lambda = 0.2$ is suitable in almost all cases.

representation of the path in that it does not represent the order in which the actions are performed. That information is not needed hereafter, so this is a sufficient representation.

Because this is a greedy technique, it is possible that the search for a path from $\mathbf{y}_i$ to $\mathbf{y}_j$ may become stuck in a local optimum. This condition is detected if more than a ratio of λ of the gap between $\mathbf{y}_i$ to $\mathbf{y}_j$ is unexplained by the estimated path. When this occurs, the path is rejected as an invalid estimate, and $\mathbf{y}_j$ is determined to not be a neighbor of $\mathbf{y}_i$. If the observation manifold exhibits local linearity, which is typically assumed, then short paths will not encounter local optima. Since long paths will be rejected anyway, there is little (if any) value in fine-tuning the value of λ. The only significant case when this check has any effect is when a very distant point begins close to a local optimum. In such cases, the estimated path length will be close to zero, and nearly all of the distance between $\mathbf{y}_i$ to $\mathbf{y}_j$ will remain unexplained by the path. Thus, a wide range of values for λ yields nearly identical results. We use $\lambda = 0.2$ in all of our experiments.

Typically NLDR uses one of two common techniques for determining local neighborhoods. The most common technique is to compute the k-nearest Euclidean distance neighbors of every $\mathbf{y}_t \in \mathbf{Y}$. A less-popular technique is to choose all points that fall within a distance of ϵ to be neighbors. This approach is less popular because it requires a problem-specific value for ϵ. A good value can be difficult to determine since distances in observation space depend on the nature of the observations. Since TNLDR uses a problem-independent time-based distance metric, however, it is possible to intuitively select a value for ϵ that will be suitable with many different problems. Each $\mathbf{y}_j$ is determined to be a neighbor of $\mathbf{y}_i$ if $||\mathbf{p}(i,j)|| \leq \epsilon$. In our implementation, we use the value $\epsilon = 2$. Intuitively, this means that some observation is determined to be a neighbor of the current observation if it can be reached by performing two or fewer actions. This creates a local neighborhood size that is suitable for most problems, and is robust even when some regions of the context space are sampled more heavily than others, which is typical with random walks.

If the actions are continuous, then $||\mathbf{p}(i,j)||$ is computed as the length of the path, or Manhattan magnitude. If the actions are discrete, then $||\mathbf{p}(i,j)||$ is the Euclidean magnitude of $\mathbf{p}(i,j)$. It is computed this way because the NLDR algorithm used in the next step will assume a continuous space while computing the context embedding, and will seek to preserve the Euclidean distance between points.

Step 3: $\mathbf{X}$ is computed by using an NLDR algorithm with the table of normalized neighbor distances, $\mathbf{D}$. Some NLDR algorithms that inherently support custom distance metrics include: Isomap [Tenenbaum et al., 2000], Local Multidimensional Scaling [Venna and Kaski, 2006], and Breadth-first Unfolding . Some other existing NLDR algorithms compute distances internally, usually using Euclidean distance, and do not explicitly support custom distance metrics. These NLDR algorithms may need to be modified somewhat to be suitable for use with TNLDR.

7.4 Experimental Validation

This section reports results from experiments designed to validate the utility of TNLDR. Section 7.4.1 compares the state estimates of TNLDR with those of the corresponding NLDR algorithms. Section 7.4.2 demonstrates the use of TNLDR to build a more accurate model of a dynamical systems than existing methods. Section 7.4.3 demonstrates that TNLDR can be used to build models of dynamical systems using recurrent versions of arbitrary regression models. Section 7.4.4 demonstrates that a model trained by TNLDR is sufficiently accurate to facilitate planning.

7.4.1 State Estimation

We used a simulated system involving a virtual crane with a boom and a ball that hangs from a cable. There were 4 actions associated with this system: {rotate right (yaw-wise), rotate left, extend the length of the cable, shorten the cable}. A ray-tracer was used to generate 64×48 pixel 3-channel observation images of this system, such that each observation was a 9216-dimensional vector. We generated a sequence of 4000 random actions, $\mathbf{A}$, and applied them to this system to obtain the corresponding 4000 observation images, $\mathbf{Y}$

Figure 7.4A shows the result of using principal component analysis, a linear dimensionality reduction technique, to reduce $\mathbf{Y}$ into two dimensions. This visualization shows that the actions have a non-uniform impact on the observations. In most cases, changing the yaw angle of the crane has a large impact on the system in observation space, while changing the length of the cable has a relatively small impact in observation space. This is manifest in the formation of small string-like clusters in the PCA plot. Each "string" is composed of observations with the same yaw-angle, but different cable lengths. Effective neighbor-finding is difficult in this space for two reasons: First, Euclidean-distance will tend to pick only neighbors with the same yaw-angle, since the cable-length has a lesser impact on observations. Second, the random walk samples the space non-uniformly. In order to facilitate NLDR, local neighborhoods must be transitively connected across the entire manifold, but in order to

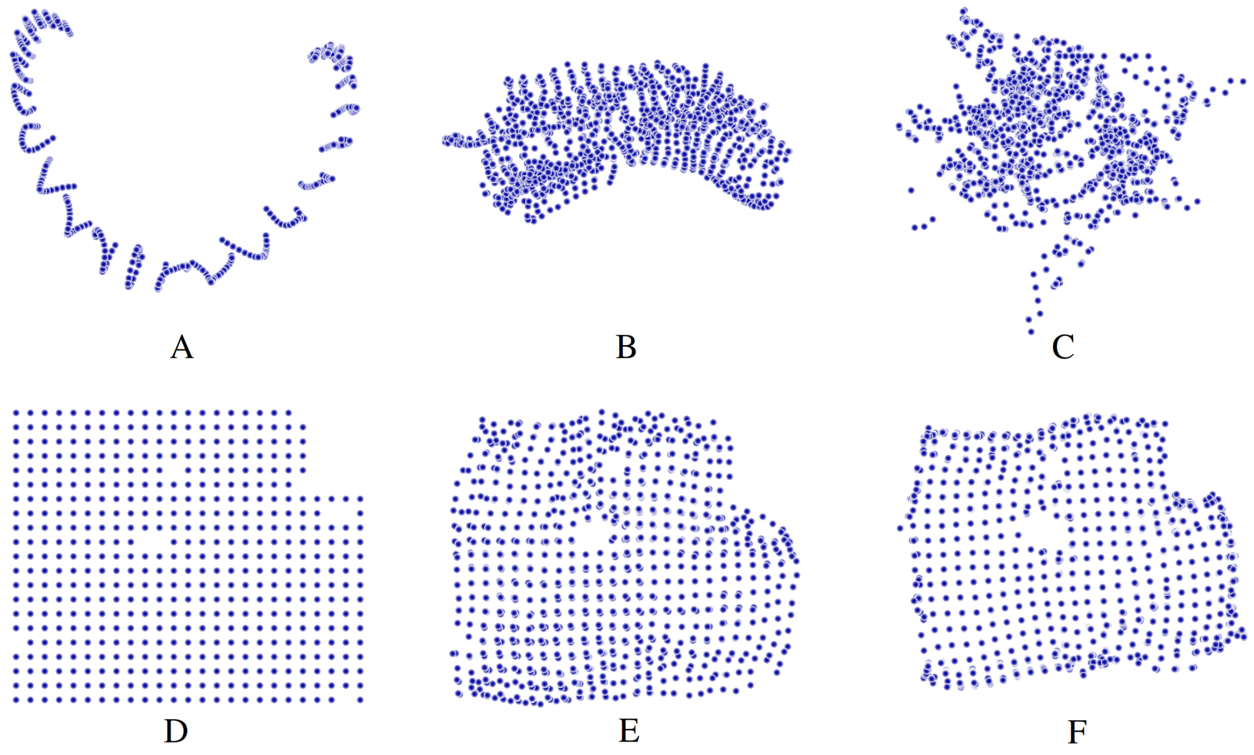


Figure 7.4: A) A PCA projection of $\mathbf{Y}$ into 2 dimensions. B) An Isomap projection of $\mathbf{Y}$. C) A Breadth-first Unfolding projection of $\mathbf{Y}$. D) The actual hidden states visited by the random walk. E) A TNLDR projection of $\mathbf{Y}$ using Isomap. F) A TNLDR projection of $\mathbf{Y}$ using Breadth-first Unfolding. TNLDR estimates the state of dynamical systems better than regular NLDR algorithms.

achieve this using Euclidean-distance, the neighborhoods must be so large that undesirable topological structures will be represented in the neighborhoods.

Figure 7.4B shows these observations reduced using Isomap with neighborhoods of size 48. The large number of neighbors was necessary to produce transitive connectivity. Figure 7.4C shows results with Breadth-first Unfolding (BFU) with neighborhoods of size 48. For comparison, Figure 7.4D shows the actual hidden state of the system. Figure 7.4E shows results with TNLDR using Isomap. Figure 7.4F shows results with TNLDR using BFU. TNLDR improved the results from both algorithms. When better distances go into an NLDR algorithm, better state estimates comes out.

7.4.2 Modeling Dynamical Systems

Perhaps the best way to demonstrate the utility of TNLDR is to demonstrate its use in training a model of a dynamical system. This is done with a simple method that we call *State Estimate Induced Training* (SEIT). The steps of SEIT are:

1. Use TNLDR to compute $\mathbf{X}$ from $\{\mathbf{Y}, \mathbf{A}\}$.
2. Train a regression model, f, with each $\langle \mathbf{a}_t, \mathbf{x}_t \rangle \rightarrow \mathbf{x}_{t+1}$, where $t < n - 1$.
3. Train a regression model, g, with each $\mathbf{x}_t \rightarrow \mathbf{y}_t$, where $t < n$.
4. Return $\langle \mathbf{x}_0, f, g \rangle$. (Note that $\mathbf{x}_0 \in \mathbf{X}$.)

SEIT uses TNLDR to divide the recurrent model shown in Figure 7.1 into two simpler parts, f and g, which may each be trained as a static model. SEIT computes a model of a dynamical system, $\langle \mathbf{x}_0, f, g \rangle$, where $\mathbf{x}_0$ is the initial estimate of state, f is a transition function which specifies how the state changes at each time-step, and g is an observation function which specifies the relationship between observations and state. If f and g are both modeled with feed-forward neural networks, then the model as a whole is an Elman recurrent neural network. SEIT provides a convenient mechanism for validating the effectiveness of TNLDR, because it may be compared empirically with existing methods for training Elman recurrent neural networks, such as Backpropagation Through Time (BPTT), evolutionary optimization, simulated annealing, etc.

In contrast with BPTT, SEIT has several advantages. BPTT must simultaneously train f and g so that these two functions will learn to cooperate. This makes it highly susceptible to problems with local optima. Also, BPTT is not suitable for training some models, such as support vector machines or regression trees. By contrast, SEIT determines how f and g should cooperate before their training begins, and encodes this information in $\mathbf{X}$. Thus, f and g can be trained independently, and they are trained against a stationary target. The NLDR component of TNLDR naturally avoids local optima in the representation of $\mathbf{X}$.

$$\begin{tikzcd}
\mathbf{a}_t \arrow[dr] & & & \mathbf{u} \arrow[dr] & & \\
\mathbf{x}_t \arrow[r] & \boxed{f} \arrow[r] & \mathbf{x}_{t+1} \arrow[ll, bend left] \arrow[r] & \boxed{g} \arrow[r] & g(\mathbf{u}, \mathbf{x}_{t+1})
\end{tikzcd}$$

Figure 7.5: High-dimensional observations may be parameterized with a vector, $\mathbf{u}$, such that the model predicts only the portion of $\mathbf{y}_{t+1}$ specified by $\mathbf{u}$. For example, if $\mathbf{y}_{t+1}$ is an image, then $g(\mathbf{u}, \mathbf{x}_{t+1})$ could be a prediction of the three channel values (red, green, and blue) of pixel $\mathbf{u}$ in that image.

Further, SEIT can be used to train arbitrary regression models to operate in a recurrent manner.

We trained a model of the crane system, $\langle \mathbf{x}_0, f, g \rangle$, using SEIT. We modeled f using a feed-forward neural network with 6 inputs (4 to represent the actions, and 2 from recurrent connections), one hidden layer of 4 sigmoid units, and 2 output units. We modeled g using a feed-forward neural network with 4 inputs (2 from f, and 2 to parameterize the image pixel as shown in Figure 7.5), and two hidden layers. The first hidden layer (in feed-forward order) contained 15 units, and the second hidden layer contained 30 units. g had 3 output units (for the three color channels). Together, f and g form a recurrent neural network with 668 weights. We tested five algorithms for training this network from $\{\mathbf{Y}, \mathbf{A}\}$. At 120-second intervals during training, we measured the root-mean-squared predictive accuracy of each model, averaged over 5 validation sequences, each one consisting of 40 random actions and the corresponding observations. Figure 7.6 shows a comparison of the results obtained by each algorithm. SEIT required almost 500 seconds to compute $\mathbf{X}$ using TNLDR, and to train f. Results for SEIT are only shown during the training of g. (With SEIT, f and g could be trained in parallel, but we did not utilize this advantage in this experiment.) Three of the algorithms all arrived at the poor solution of always predicting a completely white image. With this problem, there is a broad locally-convex region around this solution because the significant majority of the pixels in the true observation associated with every state is white. It may be that the evolutionary optimizer would eventually find its way out of this local optimum, but even if it does, this is a very inefficient solution. We ran that algorithm for 7

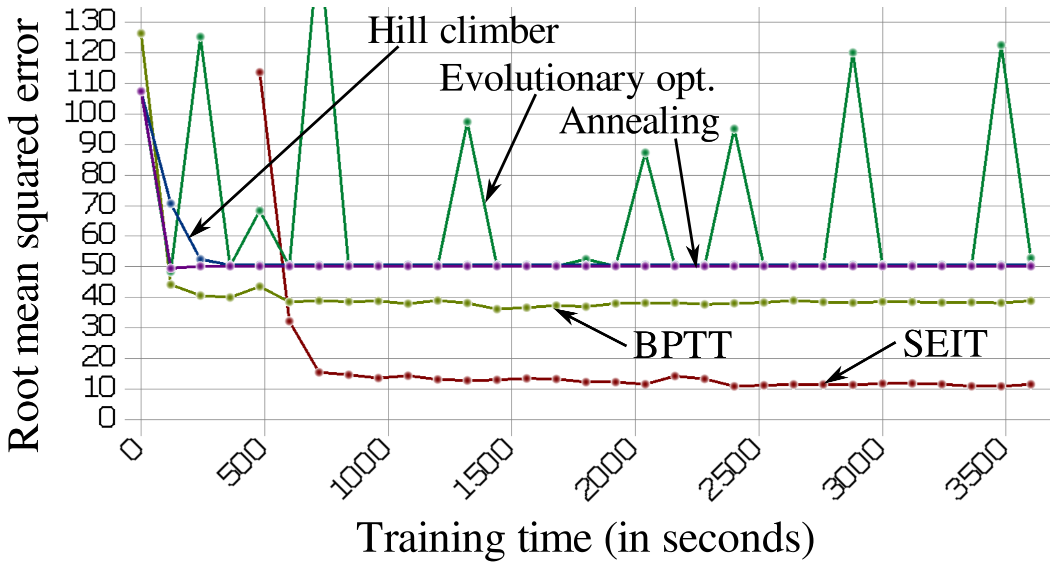


Figure 7.6: Predictive error averaged over 5 unique validation sequences, each with 40 actions and observations, was measured at 120 second intervals during training with 5 algorithms. Three algorithms converged to always predict a blank image. BPTT did better. SEIT gave the best results. Approximately the first 500 seconds were required to compute **X** and to train the transition function, so results are shown for SEIT during training of the observation function.

additional hours, but it did not manage to break out in that time. Backpropagation Through Time was the closest competitor with SEIT. Unfortunately, it appears to have quickly found a local optimum from which it never managed to escape. SEIT produced the best results by a significant margin.

Next, we modified the crane system to add random noise to each hidden state variable at every transition. The noise was drawn from a Normal distribution with a deviation equal to 5% of the magnitude of the change in state. Random noise was also added to all three channels of every pixel in the observation. This noise was also drawn from a Normal distribution with a deviation equal to 5% of the supported range in channel values. Figure 7.7(left) shows a plot of the actual hidden state of the system, and Figure 7.7(right) shows **X** as estimated by TNLDR using Breadth-first Unfolding. Despite the noise in observations, TNLDR was still able to estimate a good representation of the system state. We note that for training a

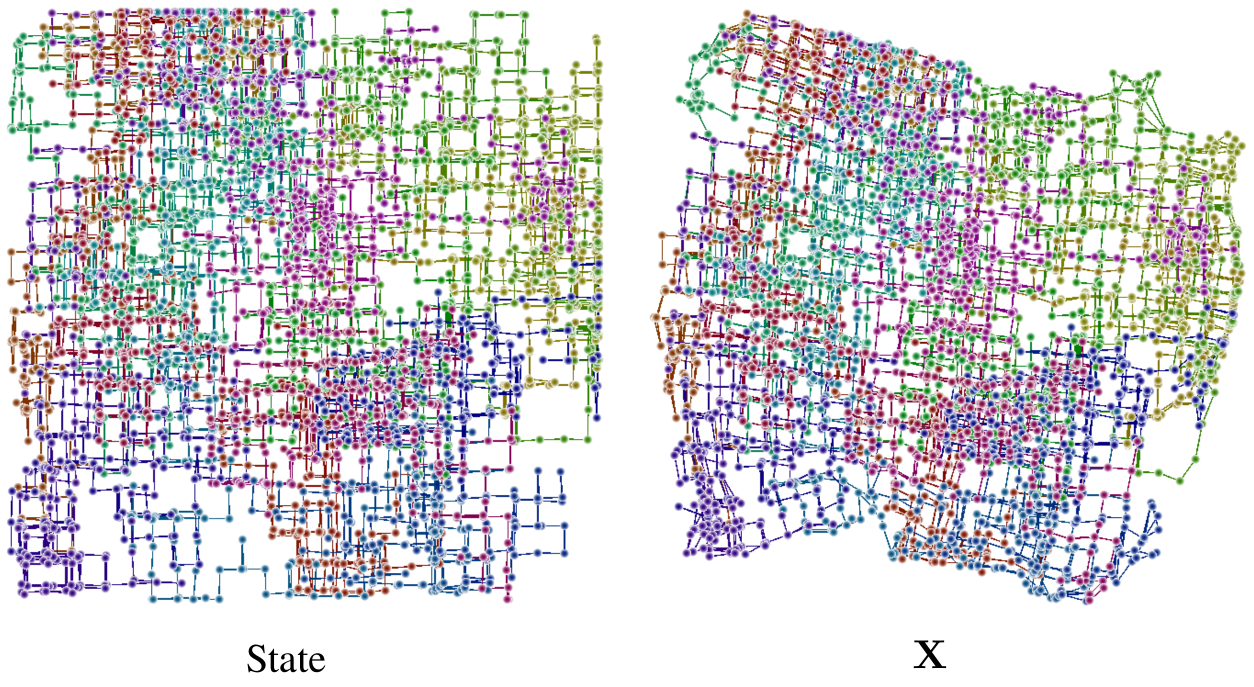


Figure 7.7: A comparison of the true hidden state from the noisy crane system, and the estimate of state, $\mathbf{X}$, computed by the first three steps of SEIT. To assist a visual comparison of the structure, each point is shown with a spectrum color according to its position in the sequence, and lines are also shown to indicate transitions.

model of the system, $\mathbf{X}$ does not need to be strictly equivalent to the hidden state, as long as f and g are able to compensate for differences.

Figure 7.8 shows a comparison of the actual observations from the noisy system, with predictions from SEIT and BPTT, over a test sequence of 200 random actions. The observation sequence was predicted from only the test actions, without any feedback from the system. SEIT predicted each frame clearly, while BPTT made blurry and ambiguous predictions. With BPTT, the interplay between f and g during training caused the internal state estimate to fall into a local optima. By contrast, SEIT did a better job of directing how f and g should mutually behave by computing $\mathbf{X}$.

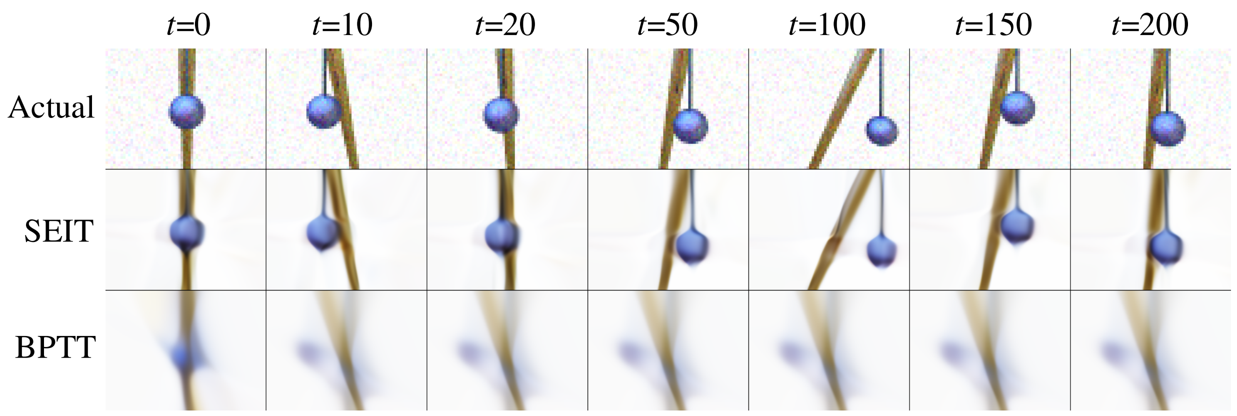


Figure 7.8: Samples of actual and predicted observations for a test sequence of actions, unrelated to **A**. Predictions were made by the recurrent models from the test actions, without any feedback from the system. SEIT made accurate and clear predictions, while BPTT made blurry predictions.

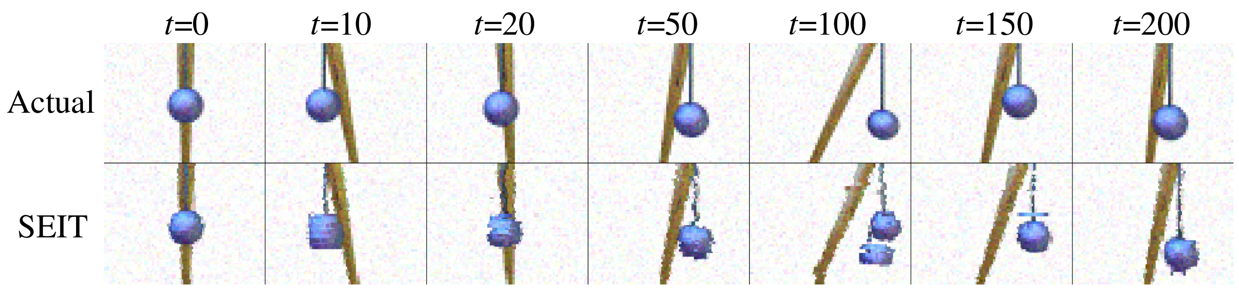


Figure 7.9: Sample predictions from a model using decision trees for f and g. Existing algorithms are only suitable for training recurrent neural networks. SEIT can train arbitrary recurrent models.

7.4.3 Decision Tree Model

In order to demonstrate that TNLDR enables the training of a recurrent version of arbitrary regression models, not just recurrent neural networks, we trained a decision tree to model both f and g. Figure 7.9 shows a comparison of actual and predicted observations with this model. Existing methods for training recurrent neural networks, such as BPTT, are not able to train models based on decision trees.

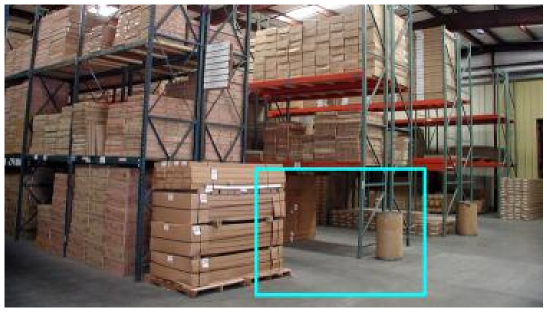


Figure 7.10: A robot's observations were simulated using a sliding and scaling window over an image of a warehouse.

7.4.4 Path Planning

We created another system using the image of a warehouse shown in Figure 7.10. The observations for this system were taken from a small window within this larger image. The system was equipped with 4 actions, where two of the actions slide the window left or right, and the other two actions zoom in or out by changing the size of the window. The window is capable of having a continuous position and size, so we used linear interpolation to generate a 64×48 pixel observation image that spans the windowed region of the larger image. The observations of this system were designed to be similar to those of a robot that navigates within a warehouse. As with the noisy crane system, we added Gaussian noise to both the transitions and observations, with the same deviations used in that system.

Additionally, we blocked the system from being able to enter a square region of its state space. A robot, for example, may be blocked by a large object from entering certain regions of its state space. Such a robot may need to learn to model its environment even though it is unable to obtain observations from those regions of its state space. We generated a new training sequence of 4000 random actions, and obtained a corresponding sequence of observations from this warehouse system. Figure 7.11 shows a comparison of the actual hidden state produced by this system, and the estimate of state computed by the first step of SEIT.

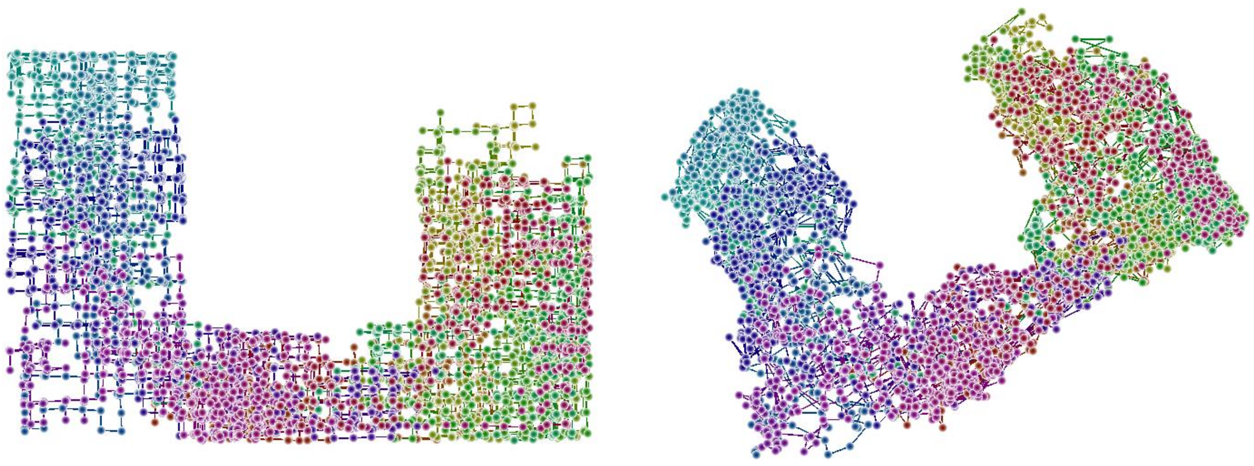


Figure 7.11: Left: A plot of the hidden states through which the warehouse system passed while generating the training observations. Right: A plot of $\mathbf{X}$ as computed by the first step of SEIT. Color is used to indicate the position of points in the ordered sequence.

We modeled this system with a recurrent neural network, where f had one hidden layer of 4 units, and two context units, and g had two hidden layers. The first hidden layer in feed-forward order had 20 units, and the second hidden layer had 100 units. We used more units in g with the warehouse system because its observations contained more detail than the crane system.

Figure 7.12 shows a comparison of results with this problem using several training algorithms. Accuracy was measured by averaging over 5 validation sequences of 40 actions and observations containing both observation noise and transition noise. The transition noise has a particularly significant impact on predictions because it accumulates in the state over time, while the observation noise affects only the current observation. Even under these conditions, SEIT was able to give the best results of any of the algorithms, using either Isomap or Breadth-first Unfolding.

Next, using only the trained neural network model to predict observations, a human oracle selected a sequence of actions that would cause the simulated robot to visit certain locations within its environment. The intrinsic states in this planned path are shown superimposed over a plot of $\mathbf{X}$ in Figure 7.13(left). We then executed the planned sequence

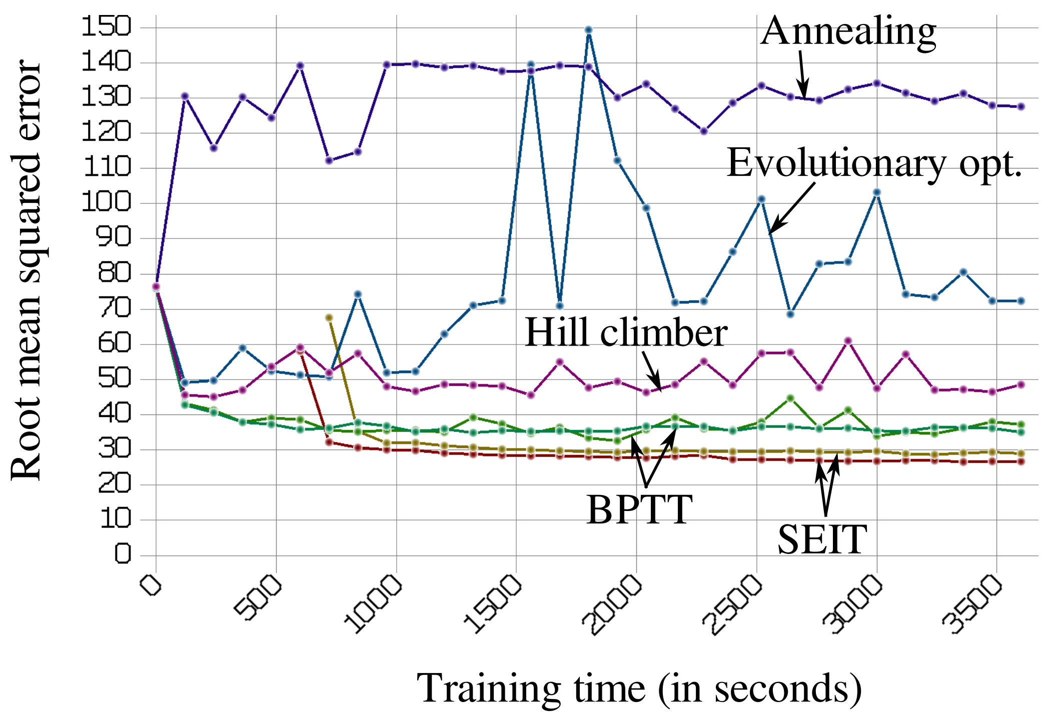


Figure 7.12: Predictive error with validation data was measured at 120 second intervals during training with several algorithms. SEIT gave the best results. The choice of NLDR algorithm used with SEIT made little difference.

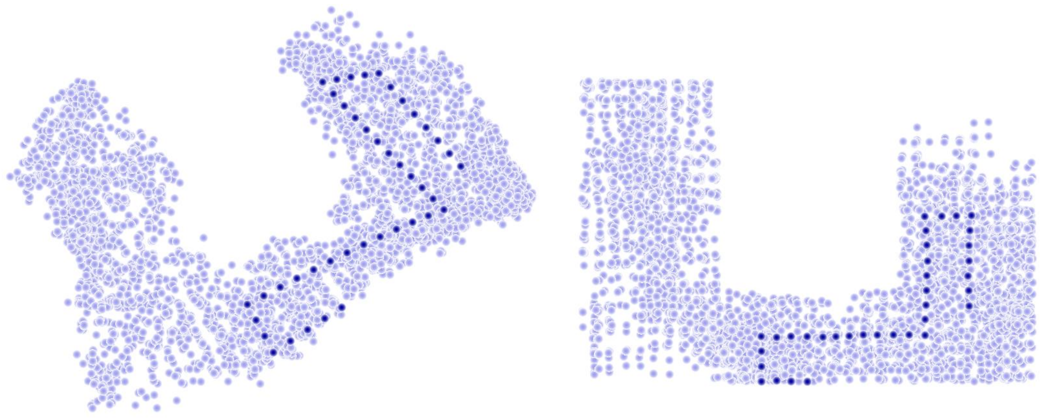


Figure 7.13: Left: A path of planned context values chosen by a human based on predicted observations from the neural network model of the warehouse system. Right: The path of actual state values through which the system passed when the planned actions were applied to the system.

of actions with the actual system. Figure 7.13(right) shows the actual hidden state values through which the system passed as it followed the sequence of actions, superimposed over

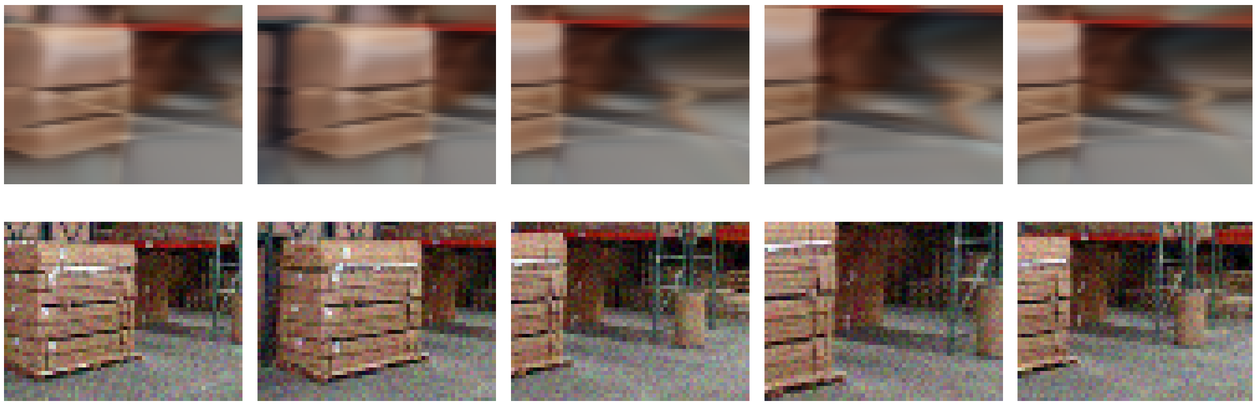


Figure 7.14: Top: Predicted observations from a planned path made using only a model of the warehouse system. Bottom: The actual observations made when the planned path was executed with the warehouse system.

the actual state values that correspond with the training observations that were originally used to train the model. Figure 7.14(top) shows predicted observations at five points along the planned path. Figure 7.14(bottom) shows the actual observations at those points when the plan was executed on the system. This experiment demonstrates that the trained model represented the system with sufficient accuracy that it could facilitate planning in isolation from the system.

7.5 Conclusions

We presented a new technique called TNLDR, which reduces the dimensionality of observations from a dynamical system to recover an estimate of the system state. Compared with regular NLDR, TNLDR uses the additional information found in sequences of observations to remove the assumption that distances in state space are proportional to distances in observation space. Because TNLDR removes this assumption, it can compute accurate estimates of state, even when various factors cause observations to be scaled non-uniformly. TNLDR has significant potential to lead to further innovations because it extends existing NLDR techniques to make them suitable for use in estimating the state of dynamical systems from high-dimensional observations.

We used a simulated crane system to demonstrate that TNLDR recovers better estimates of state than existing NLDR techniques. We also demonstrated that TNLDR leads to a natural method for training models of dynamical systems, called SEIT. We showed that SEIT does a better job training a recurrent neural network to model the crane system than existing methods for training recurrent neural networks. We also repeated this experiment using a system involving a simulated robot in a warehouse. We showed that SEIT is suitable for training other recurrent models besides neural networks. We also demonstrated that models trained by SEIT are sufficiently accurate to facilitate planning.

7.5.1 Appendix

This section contains information that was omitted from the published version of this paper/chapter due to space constraints imposed by the venue. Due to the significance of the results presented in this section, we include it here as part of this dissertation.

In this section, we report an experiment designed to test the ability of the model trained with SEIT to generalize about states that were not visited in the training sequence. This experiment was not performed with the other algorithms because none of them were able to produce sufficiently comprehensible predictions.

We created a test sequence of 48 actions that follow the path indicated in Figure 7.15. 12 of the 48 states on this path were never visited by the training sequence, and 22 of the 48 state-action pairs on this path were never sampled in the training sequence. We executed this sequence of actions with the crane system, and with the model trained by SEIT. Calibration was not used, so the model received no feedback from the system during this execution. Six frames at significant locations on this path are shown in Figure 7.16.

The prediction at $t = 0$ is only a measure of how well the neural network g can be trained to represent the image, since f has not been used at this point at all. The accuracy

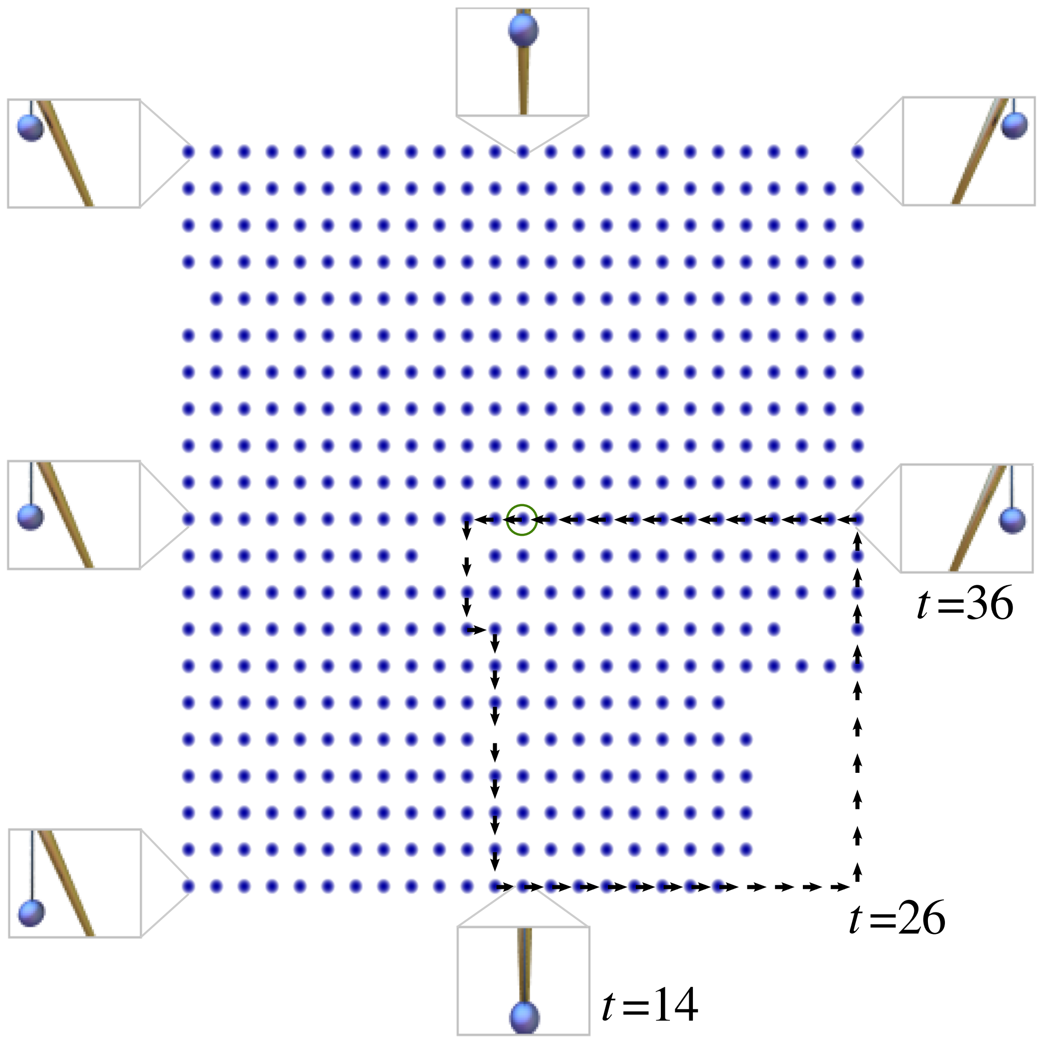


Figure 7.15: A test sequence of 48 actions was designed to pass through regions of the state-space that were not visited by the training sequence. This sequence begins and ends with the circled state in the center. 12 of the 48 states on this path were never visited by the training sequence, and 22 of the 48 state-action pairs on this path were never sampled in the training sequence. To establish context, 7 points are labeled with the corresponding observation image from the system.

of subsequent frames, however, depends on the accuracy of both f and g. With this path of actions, the actual system begins and ends in the same state. Note that the target frame at $t = 0$ is the same as at $t = 48$. The prediction at $t = 48$, however, shows the ball hanging slightly too low and slightly off-center. This indicates the extent of the accumulated error in f over the entire path. The reason f is able to generalize so effectively for state-actions that were not in the training sequence is because SEIT was able to arrange the context estimates, $\mathbf{X}$, in a nearly grid-like arrangement, even though the observations fall on a very non-linear manifold. The nearly-linear arrangement of context estimates is much more conducive to accurate generalization.

The generalizing ability of g is tested the most in the predictions near $t = 26$. Since the model finished this path (at $t = 48$) with a predicted observation so close to that of the actual system, it is reasonable to assume that the context estimate is somewhat accurate along the entire path. The error between the target and predicted observation at $t = 26$, therefore, is mostly due to g, rather than to f. Although there is some obvious distortion in the prediction at $t = 26$ (for example, the cable does not hang in a perfectly straight line), we note that this prediction is sufficient to characterize the state of the system. In other words, this model is sufficiently accurate for planning purposes. We also note that the predicted observations become sharp again when the system returns to regions of the state-space with which it is familiar.

When the observations are high-dimensional, models that use observations for their estimate of state, rather than computing their own internal representation of context, are not as well-suited for planning because they are not able to generalize in this manner. Even slight imprecisions in the model will cause the predictions to drift away from the manifold of meaningful observations. When an internal estimate of context is used in the model, however, the dimensionality of that context-space can be limited to match the dimensionality of the state space of the system. Thus, all possible context vectors have a corresponding state, and the model can be used for making plans into the distant future.

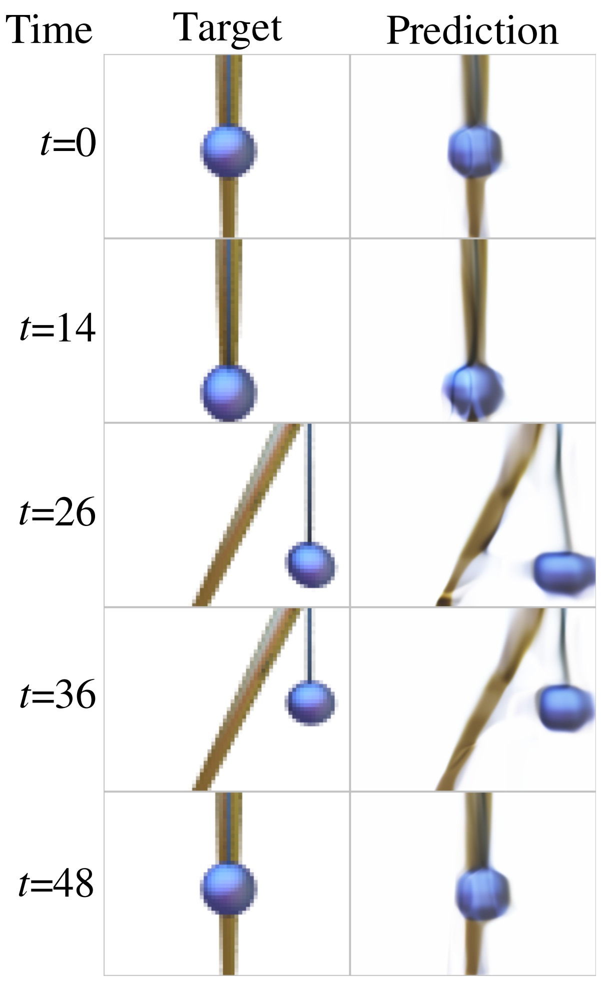


Figure 7.16: Target and predicted observations at significant time-steps along the human-generated path of test actions that was designed to pass through regions of the space that were not sampled in the training sequence.

Chapter 8

Missing Value Imputation With Unsupervised Backpropagation

Abstract: Many data mining and data analysis techniques operate on dense matrices or complete tables of data. Real-world data sets, however, often contain unknown values. Even many classification algorithms that are designed to operate with missing values still exhibit deteriorated accuracy. One approach to handling missing values values is to fill in (impute) the missing values. In this paper, we present a technique for unsupervised learning called *Unsupervised Backpropagation* (UBP), which trains a multi-layer perceptron to fit to the manifold sampled by a set of observed point-vectors. We evaluate UBP with the task of imputing missing values in datasets, and show that UBP is able to predict missing values with significantly lower sum-squared error than other collaborative filtering and imputation techniques. We also demonstrate with 31 datasets and 9 supervised learning algorithms that classification accuracy is usually higher when randomly-withheld values are imputed using UBP, rather than with other methods.

8.1 Introduction

Many effective machine learning techniques are designed to operate on dense matrices or complete tables of data. Unfortunately, real-world datasets often include only samples of observed values mixed with many missing or unknown elements. Missing values may occur due to human impatience, human error during data entry, data loss, faulty sensory equipment, changes in data collection methods, inability to decipher handwriting, privacy issues, legal requirements, and a variety of other practical factors. Thus, improvements to methods for imputing missing values can have far-reaching impact on improving the effectiveness of existing learning algorithms for operating on real-world data. We present a method for imputation called *Unsupervised Backpropagation* (UBP), which trains a multi-layer perceptron (MLP) to fit to the manifold represented by the known features in a dataset. We demonstrate this algorithm with the task of imputing missing completely at random (MCAR) values, and we show that it is significantly more effective than other methods at this task.

Backpropagation has long been a popular method for training neural networks [Rumelhart et al., 1986, Werbos, 1990]. A typical supervised approach trains the weights, $\mathbf{W}$, of a multilayer perceptron (MLP) to fit to some training data, consisting of a set of feature vectors $\mathbf{X} = \langle \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \rangle$, and corresponding label vectors $\mathbf{Y} = \langle \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n \rangle$. With many interesting problems, however, training data is not available in this form. In this paper, we consider the significantly different problem of training an MLP to estimate the missing elements of an $n \times d$ matrix, $\mathbf{X}$, where each of the d attributes may be continuous or categorical. Because the missing elements in $\mathbf{X}$ must be predicted, it becomes the output of the MLP, rather than the input. A new set of latent vectors, $\mathbf{V} = \langle \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n \rangle$, will be fed as inputs into the MLP, but no examples from $\mathbf{V}$ are given in the training data. Thus, both $\mathbf{V}$ and $\mathbf{W}$ must be trained using only the known elements in $\mathbf{X}$. After training, each $\mathbf{v}_i$ may be fed into the MLP to predict all of the elements in $\mathbf{x}_i$.

Training in this manner causes the MLP to fit a surface to the (typically non-linear) manifold sampled by $\mathbf{X}$. After training, $\mathbf{V}$ may be considered to be a reduced-dimensional representation of $\mathbf{X}$. That is, $\mathbf{V}$ will be an $n \times t$ matrix, where $t < d$, and the MLP will map $\mathbf{V} \mapsto \mathbf{X}$.

UBP accomplishes the task of training an MLP using only the known elements in $\mathbf{X}$ with on-line backpropagation. For each presentation of a known element $x_{r,c} \in \mathbf{X}$, UBP simultaneously computes a gradient vector $\mathbf{g}$ to update the weights $\mathbf{W}$, and a gradient vector $\mathbf{h}$ to update the input vector $\mathbf{v}_r$. ($x_{r,c}$ is the element in row r, column c of $\mathbf{X}$.)

As an algorithm, UBP falls at the intersection of several different paradigms. In neural networks, it is an extension of generative backpropagation [Hinton, 1988] that does not require the weights to be trained in advance. In collaborative filtering, it may be considered to be a non-linear generalization of matrix factorization [Koren et al., 2009]. As an imputation technique, UBP is a refinement of Non-linear PCA [Scholz et al., 2005]. Because UBP generates $\mathbf{V}$ from $\mathbf{X}$, it is a non-linear dimensionality reduction algorithm, and is also comparable in this paradigm with Non-linear PCA. Because it trains a generalizing MLP to model the manifold sampled by $\mathbf{X}$, it is also a manifold learning algorithm, and may be considered to perform a similar task to the latter half of an autoencoder. In Section 8.2, we further describe the relationship of UBP with each of these existing techniques. In this paper, we demonstrate UBP as a method for imputing missing values, and show that it outperforms other approaches at this task. We will defer to future papers to demonstrate UBP with other applications.

We compare UBP against 5 other imputation methods on a set of 31 data sets. 10% to 90% of the values are removed from the data sets completely at random. We show that UBP predicts the missing values with signficantly lower error (as measured by sum-squared difference with normalized values) than other approaches. We also evaluated 9 learning algorithms to compare classification accuracy using imputed data sets. Learning algorithms using imputed data from UBP usually achieve higher classification accuracy than with any of the other methods. The increase is most significant when 30% to 70% of the data is missing.

The remainder of this paper is organized as follows. Section 8.2 reviews related work to UBP and missing value imputation. UBP is described in Section 8.3. Section 8.4 presents the results of comparing UBP with other imputation methods. We provide conclusions and a discussion of future directions for UBP in Section 8.5.

8.2 Related Works

The techniques used in UBP were first proposed by Hinton [Hinton, 1988] in a technique he called "generative backpropagation." Generative backpropagation adjusts the inputs in a neural network while holding the weights constant. UBP, by contrast, computes both the weights and the input values simultaneously. Related approaches have been used to generate labels for images [Coheh and Shawe-Taylor, 1990], and for natural language [Bengio et al., 2006]. Although these techniques have been used for labeling images and documents, to our knowledge, they have not been used for the application of imputing missing values. UBP differs from generative backpropagation in that it trains the weights simultaneously with the inputs, instead of training them as a pre-processing step.

UBP may also be classified as a manifold learning algorithm. Like common non-linear dimensionality reduction (NLDR) algorithms, such as Isomap [Tenenbaum et al., 2000], MLLE [Zhang and Wang, 2007], or Manifold Sculpting [Gashler et al., 2008b], it reduces a set of high-dimensional vectors, $\mathbf{X}$, to a corresponding set of low-dimensional vectors, $\mathbf{V}$. Unlike these algorithms, however, UBP also learns a model of the manifold. Also unlike these algorithms, UBP is designed to operate with incomplete observations.

UBP may be viewed as a non-linear generalization of matrix factorization (MF). MF is a linear dimensionality reduction technique that can be effective for collaborative filtering [Adomavicius and Tuzhilin, 2005] as well as imputation. This method has become a popular technique, in part due to its effectiveness with the data used in the NetFlix competition [Koren et al., 2009]. MF involves factoring the data matrix into two much-smaller matrices. These smaller matrices can then be combined to predict all of the missing values in the original dataset. It is equivalent to using linear regression to project the data onto its first few principal components. Unfortunately, MF is not well-suited for data that exhibits non-linearities. It was previously shown that matrix factorization could be represented with a neural network model involving one hidden layer and linear activation functions [Takács et al., 2009]. In comparison with this approach, UBP uses a standard MLP with an arbitrary number of hidden layers and non-linear activation functions, instead of the network structure previously proposed for matrix factorization. MF produces very good results at the task of imputation, but we demonstrate that UBP does better.

UBP is very similar to an existing method called Nonlinear PCA [Scholz et al., 2005] (NLPCA), which has been shown to be effective for imputation. This approach also uses gradient descent to train an MLP to map from low to high-dimensional space. After training, the weights of the MLP can be used to represent non-linear components within the data. If these components are extracted one-at-a-time from the data, then they are the principal components, and NLPCA becomes a non-linear generalization of PCA. Typically, however, these components are all learned together, so it would more properly be termed a non-linear generalization of MF. NLPCA was evaluated with the task of missing value imputation [Scholz et al., 2005], but its relationship to MF was not yet recognized at the time, so it was not compared against MF. One of the contributions of this paper is that we show NLPCA to be a significant improvement over MF at the task of imputation. We also demonstrate that UBP achieves even better results than NLPCA at the same task, and is the best algorithm for imputation of which we are aware. The primary difference between NLPCA and UBP is that UBP utilizes a three-phase training approach (described in Section 8.3) which makes it more robust against falling into a local optimum during training.

UBP is also comparable with the latter-half of an autoencoder [Hinton and Salakhutdinov, 2006]. The first half of an autoencoder maps data into low-dimensional space. The second half of an autoencoder maps the data back to the original space. By comparison, UBP trains an MLP to map data from low to high-dimensional space. Because each layer in an MLP adds an increasingly expensive cost in training time, and because UBP trains a network with half the depth of a corresponding autoencoder, UBP is practical for many problems for which autoencoders are too computationally expensive.

Since we demonstrate UBP with the application of imputing MCAR values in data, it is also relevant to consider other approaches that are classically used for this task. Simple methods, such as dropping patterns that contain missing values or randomly drawing values to replace the missing values, are often used based on simplicity for implementation. These methods, however, have significant obvious disadvantages when data is scarce. Another common approach is to treat missing elements as having a unique value. This approach, however has been shown to bias the parameter estimates for multiple linear regression models [Jones, 1996] and to cause problems for inference with many models [Shafer, 1997]. We take it for granted that better accuracy is desirable, so these methods should generally not be used, as better methods do exist. In order to establish a "baseline" for comparison, we compare with the method of replacing missing values in continuous attributes with the mean of the non-missing values in that attribute, and replacing missing values in nominal (or categorical) attributes with the most common value in the non-missing values of that attribute. It is expected that any reasonable algorithm should outperform this baseline (BL) algorithm with most problems.

A simple improvement over BL is to compute a separate centroid for each output class. The disadvantages of this method are that it is not suitable for regression problems, and it cannot generalize to unlabeled data since it depends on labels to impute. Methods based on maximum likelihood [Little and Rubin, 2002] have long been studied in statistics, but these also depend on pattern labels. We restrict our analysis to methods that can generalize, and therefore must not depend on having labeled data.

Another well-studied approach involves training a supervised learning algorithm to predict missing values using the non-missing values as inputs [Quinlan, 1989, Lakshminarayan et al., 1996, Farhangfar et al., 2008]. Unfortunately, the case where multiple values are missing in one pattern present a difficulty for these approaches. Either a learning algorithm must be used that implicitly handles missing values in some manner, or an exponential number of models must be trained to handle each combination of missing values. Further, it has also been shown that results with these methods tend to be poor when there are high percentages (more than about 15%) of missing values [Acuña and Rodriguez, 2004].

One very effective collaborative filtering method for imputation is to cluster the data, and then make predictions according to the centroid of the cluster in which each point falls. [Adomavicius and Tuzhilin, 2005] Luengo compared several imputation methods by evaluating their effect on classification accuracy [Luengo, 2011]. He found cluster-based imputation with Fuzzy k-Means (FKM) [Li et al., 2004] using Manhattan distance to outperform other methods, including those involving state of the art machine learning methods and other methods traditionally used for imputation. Our analysis, however, finds that most of the methods we compared outperform FKM.

A related imputation method called instance-based imputation (IBI) is to combine the non-missing values of the k-nearest neighbors of a point to replace its missing values. To evaluate the similarity between points, cosine correlation is often used because it tends to be effective in the presence of missing values [Adomavicius and Tuzhilin, 2005, Li et al., 2008, Sarwar et al., 2001].

Ensemble techniques, such as multiple imputation, have also been shown to be effective for imputing missing values [Schafer and Graham, 2002]. In this paper, we do not compare against ensemble methods because UBP involves a single model, and it may be included in an ensemble as well as any other imputation method.

8.3 Unsupervised Backpropagation

In order to formally describe the UBP algorithm, we define the following terms. The relationships between these terms are illustrated graphically in Figure 8.1.

1. Let $\mathbf{X}$ be a given $n \times d$ matrix, which may have many missing elements. We seek to impute values for these elements. n is the number of instances. d is the number of attributes.
2. Let $\mathbf{V}$ be a latent $n \times t$ matrix, where $t < d$.
3. If $x_{r,c}$ is the element at row r, column c in $\mathbf{X}$, then $\hat{x}_{r,c}$ is the value predicted by the MLP for this element when $\mathbf{v}_r \in \mathbf{V}$ is fed forward into the MLP.

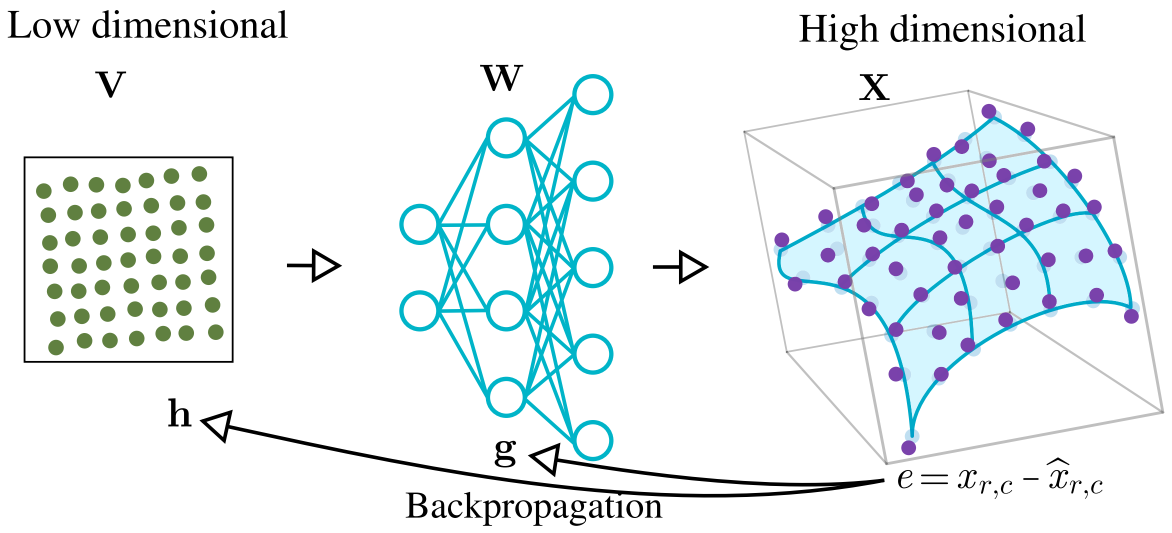


Figure 8.1: UBP trains an MLP to fit to high-dimensional observations, $\mathbf{X}$. For each known $\mathbf{x}_{r,c} \in \mathbf{X}$, it uses backpropagation to compute the gradient vectors $\mathbf{g}$ and $\mathbf{h}$, which are used to update the weights, $\mathbf{W}$, and the input vector $\mathbf{v}_r$.

4. Let w_{ij} be the weight that feeds from unit i to unit j in the MLP.
5. For each network unit, i, let β_i be the net input into the unit, let α_i be the output or activation value of the unit, and let e_i be an error term associated with the unit.
6. Let f be the activation function used in every unit, and let $f'(\beta_i)$ be the derivative of f with respect to β_i. In our implementation, we use the logistic function for f.
7. Let l be the number of hidden layers in the MLP.
8. Let $\mathbf{g}$ be a vector representing the gradient with respect to the weights of an MLP, such that $g_{i,j}$ is the component of the gradient that is used to refine $w_{i,j}$.
9. Let $\mathbf{h}$ be a vector representing the gradient with respect to the inputs of an MLP, such that h_i is the component of the gradient that is used to refine $v_{r,i} \in \mathbf{v}_r$.

Using backpropagation to compute $\mathbf{g}$, the gradient with respect to the weights, is a common operation for training MLPs [Rumelhart et al., 1986, Werbos, 1990]. Using backpropagation to compute $\mathbf{h}$, the gradient with respect to the inputs, however, is much less common, so we provide a derivation of it here. In this deriviation, we compute each $h_i \in \mathbf{h}$ from the presentation of a single element $x_{r,c} \in \mathbf{X}$. It could also be derived from the presentation of a full row (which is typically called "on-line training"), or from the presentation of all of $\mathbf{X}$ ("batch training"), but since we assume that $\mathbf{X}$ is high-dimensional and is missing many values, it is more efficient to train with the presentation of each known element individually. We begin by expressing the partial derivative of the error signal with respect to the inputs as given in Equation 8.1. We assume that sum-squared error is used for this objective function, although other objective functions could be used as well.

$$h_i = \frac{\partial \frac{1}{2}(x_{r,c} - \hat{x}_{r,c})^2}{\partial v_{r,i}} \tag{8.1}$$

By applying the chain rule to Equation 8.1 and then expanding, we obtain

$$h_i = \frac{-(x_{r,c} - \hat{x}_{r,c}) f'(\alpha_c) \sum_j w_{j,c} \partial \beta_j}{\partial v_{r,i}}. \tag{8.2}$$

If $l = 0$, then $\beta_j \equiv v_{r,i}$, so the partial derivative terms in the numerator and denominator cancel out. The error term e_c is then substituted into the equation to simplify it to

$$h_i = -w_{i,c} e_c. \tag{8.3}$$

If $l > 0$, then β_j can be expanded to obtain

$$h_i = \frac{-(x_{r,c} - \hat{x}_{r,c}) f'(\alpha_c) \sum_j w_{j,c} \partial f'(\alpha_j) \sum_k w_{k,j} \partial \beta_k}{\partial v_{r,i}}. \tag{8.4}$$

If $l = 1$, then $\beta_k \equiv v_{r,i}$. If $l > 1$, then β_k can be further expanded in the same manner until the partial derivative term in the numerator cancels out the denominator. At that point, the error terms for the earliest (adjacent to the inputs) hidden layer are plugged into the equation to simplify it to

$$h_i = -\sum_j w_{i,j} e_j. \tag{8.5}$$

8.3.1 3-phase Training

UBP trains $\mathbf{V}$ and $\mathbf{W}$ in three phases. The first phase computes an initial estimate for $\mathbf{V}$. The second phase computes an initial estimate for $\mathbf{W}$. The third phase refines them both together. All three phases train using an approach derived from backpropagation. That is, they compute an error term for each output and hidden unit in the MLP, then adjust $\mathbf{V}$ and/or $\mathbf{W}$ by a small amount in the opposite direction of the error surface gradient. We now briefly give an intuitive justification for this approach. In our initial experimentation, we used the simpler approach of training in a single phase. With several problems, we observed that early during training, the intrinsic point vectors, $\mathbf{v}_i \in \mathbf{V}$, tended to separate into clusters. The points in each cluster appeared to be unrelated, as if they were arbitrarily assigned to one of the clusters by their random initialization. As training continued, the MLP effectively created a separate mapping for each cluster in the intrinsic representation to the corresponding values in $\mathbf{X}$. While it is technically possible for an MLP to operate effectively under these conditions, it seems natural that results would be better if the points in $\mathbf{V}$ were not separated into clusters. This is achieved through 3-phase training by initializing $\mathbf{V}$ and $\mathbf{W}$ separately before refining them together.

8.3.2 Algorithm Description

Pseudo-code for the UBP algorithm, which trains $\mathbf{V}$ and $\mathbf{W}$ in three phases, is given in Algorithm 1. This algorithm calls Algorithm 2, which performs a single epoch of training. A detailed description of Algorithm 1 follows.

A matrix containing the known data values, $\mathbf{X}$, is passed in to UBP (See Algorithm 1). It returns $\mathbf{V}$ and $\mathbf{W}$. $\mathbf{V}$ is matrix such that each row, $\mathbf{v}_i$, is a low-dimensional representation of the corresponding row, $\mathbf{x}_i$. $\mathbf{W}$ is a set or ragged matrix containing weight values for an MLP that maps from each $\mathbf{v}_i$ to an approximation of $\mathbf{x}_i \in \mathbf{X}$. $\mathbf{v}_i$ may be forward-propagated into this MLP to estimate values for any missing elements in $\mathbf{x}_i$.

Algorithm 1 UBP($\mathbf{X}$)

```
1: Initialize each element in $\mathbf{V}$ with small random values.
2: Let $\mathbf{T}$ be the weights of a single-layer perceptron
3: Initialize each element in $\mathbf{T}$ with small random values.
4: $\alpha \leftarrow 0.01$; $\beta \leftarrow 0.0001$; $\gamma \leftarrow 0.00001$; $\lambda \leftarrow 0.0001$
5: $\eta \leftarrow \alpha$; $s' \leftarrow \infty$                    Phase 1: Compute initial estimate for $\mathbf{V}$.
6: while $\eta > \beta$ do
7:     $s \leftarrow$ train_epoch($\mathbf{X}, \mathbf{T}, \lambda,$ true$, 0$)
8:     if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
9:     $s' \leftarrow s$
10: end while
11: Let $\mathbf{W}$ be the weights of a multi-layer perceptron with $l$ hidden layers, $l \geq 0$
12: Initialize each element in $\mathbf{W}$ with small random values.
13: $\eta \leftarrow \alpha$; $s' \leftarrow \infty$                    Phase 2: Compute initial estimate for $\mathbf{W}$.
14: while $\eta > \beta$ do
15:     $s \leftarrow$ train_epoch($\mathbf{X}, \mathbf{W}, \lambda,$ false$, l$)
16:     if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
17:     $s' \leftarrow s$
18: end while
19: $\eta \leftarrow \alpha$; $s' \leftarrow \infty$                    Phase 3: Refine $\mathbf{V}$ and $\mathbf{W}$ together.
20: while $\eta > \beta$ do
21:     $s \leftarrow$ train_epoch($\mathbf{X}, \mathbf{W}, 0,$ true$, l$)
22:     if $1 - s/s' < \gamma$ then $\eta \leftarrow \eta/2$
23:     $s' \leftarrow s$
24: end while
25: return $\{\mathbf{V}, \mathbf{W}\}$
```

Lines 1-9 perform the first phase of training, which computes an initial estimate for $\mathbf{V}$. **Line 1** of Algorithm 1 initializes the elements in $\mathbf{V}$ with small random values. Our implementation draws values from a Normal distribution with a mean of 0 and a deviation of 0.01. **Lines 2-3** initialize the weights, $\mathbf{T}$ of a single-layer perceptron using the same mechanism. This single-layer perceptron is a temporary model that is only used in phase 1 to assist the initial training of $\mathbf{V}$. **Line 4** sets some default parameter values. α specifies an initial learning rate. The learning rate is decayed during training, such that convergence is detected when it falls below β. γ specifies the portion of improvement that is expected after each epoch, or else the learning rate is decayed. λ is a regularization term that is used during the first two phases to ensure that the weights do not become excessively saturated

Algorithm 2 train_epoch($\mathbf{X}, \mathbf{W}, \lambda, p, l$)

```
1: for each known $x_{r,c} \in \mathbf{X}$ in random order do
2:     Compute $\alpha_c$ by forward-propagating $\mathbf{v}_r$ into an MLP with weights $\mathbf{W}$.
3:     $e_c \leftarrow (x_{r,c} - \alpha_c) f'(\beta_c)$
4:     for each hidden unit $i$ feeding into output unit $c$ do
5:         $e_i \leftarrow w_{i,c} e_c f'(\beta_i)$
6:     end for
7:     for each hidden unit $j$ in an earlier hidden layer (in backward order) do
8:         $e_j \leftarrow \sum_k w_{j,k} e_k f'(\beta_j)$
9:     end for
10:    for each $w_{i,j} \in \mathbf{W}$ do
11:        $g_{i,j} \leftarrow -e_j \alpha_i$
12:    end for
13:    $\mathbf{W} \leftarrow \mathbf{W} - \eta(\mathbf{g} + \lambda \mathbf{W})$
14:    if $p$ = true then
15:        for $i$ from $0$ to $t-1$ do
16:            if $l = 0$ then $h_i \leftarrow -w_{i,c} e_c$ else $h_i \leftarrow -\sum_j w_{i,j} e_j$
17:        end for
18:        $\mathbf{v}_r \leftarrow \mathbf{v}_r - \eta(\mathbf{h} + \lambda \mathbf{v}_r)$
19:    end if
20: end for
21: $s \leftarrow$ measure RMSE with $\mathbf{X}$
22: return $s$
```

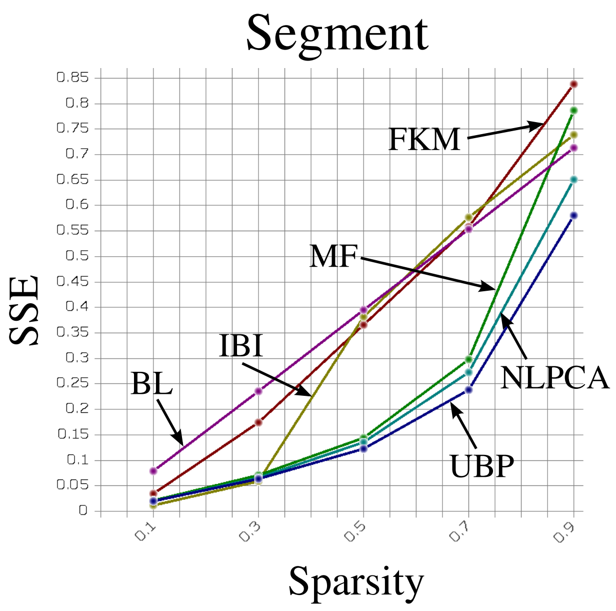
before the final phase of training. No regularization is used in the final phase of training because we want the MLP to ultimately fit the data as closely as possible. (Overfit can still be mitigated by limiting the number of hidden units used in the MLP.) Although minor adjustments to these parameters will proportionally affect when convergence is detected, we found this to have little influence on the quality of results, so we used the default values in all of our reported results. **Line 5** sets the learning rate, η, to the initial value. The value s' is used to store the previous error score. As no error has yet been measured, it is initialized to ∞. **Lines 6-9** train $\mathbf{V}$ and $\mathbf{T}$ until convergence is detected. $\mathbf{T}$ may then be discarded. **Lines 10-16** perform the second phase of training. This phase differs from the first phase in two ways: 1) The MLP is used instead of a temporary single-layer perceptron, and 2) $\mathbf{V}$ is held constant during this phase. **Lines 17-21** perform the third phase of training. In this phase, the same MLP is used again, but $\mathbf{V}$ and $\mathbf{W}$ are both refined together. Also, no regularization is used in the third phase.

Next, we describe Algorithm 2, which performs a single epoch of training. This algorithm is very similar to an epoch of traditional backpropagation, except that it presents each element individually, and it conditionally refines the values in $\mathbf{V}$ as well as the values in $\mathbf{W}$. **Line 1** presents each known element $x_{r,c} \in \mathbf{X}$ in random order. **Line 2** computes a predicted value for the presented element given the current $\mathbf{v}_r$. Note that efficient implementations of line 2 should only propagate values into output unit c. **Lines 3-7** compute an error term for output unit c, and each hidden unit in the network. The activation of the other output units is not computed, so the error on those units is 0. **Lines 8-10** refine $\mathbf{W}$ by gradient descent. **Line 11** specifies that $\mathbf{V}$ should only be refined during phases 1 and 3. **Lines 12-14** refine $\mathbf{V}$ by gradient descent. **Line 15** computes the root-mean-squared-error of the MLP for each known element in $\mathbf{X}$.

In order to enable UBP to process nominal (categorical) attributes, we convert such values to a vector representing a membership weight in each category. For example, a given value of "cat" from the value set {"mouse","cat","dog"} is represented with the vector in $\langle 0, 1, 0 \rangle$. Unknown values in this attribute are converted to 3 unknown real values, requiring the algorithm to make 3 predictions. After missing values are imputed, we convert the data back to its original form by finding the mode of each categorical distribution. For example, the predicted vector $\langle 0.4, 0.25, 0.35 \rangle$ would be converted to a prediction of "mouse".

8.4 Empirical Validation

In order to evaluate the effectiveness of UBP and related imputation techniques, we gathered a set of 31 common datasets: {arrhythmia, breast-w, bupa, cars, colic, credit-g, diabetes, ecoli, glass, heart-statlog, hepatitis, hypothyroid, ionosphere, iris, labor, letter, magic_telescope, mushroom, nursery, ozone, primary-tumor, segment, sonar, spambase, spectrometer, teaching_assistant, vehicle, vote, vowel, wine, and yeast}. To ensure an objective evaluation,



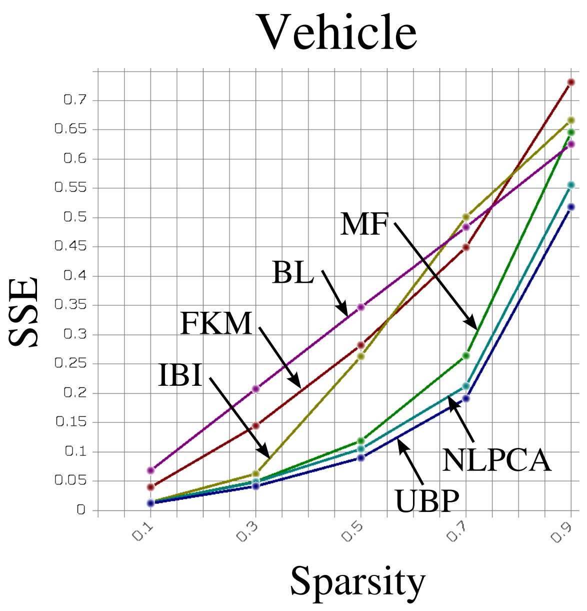


Figure 8.2: A comparison of the average sum-squared error in each pattern by 5 imputation techniques over a range of sparsity values with two representative datasets. (Lower is better.) The trends exhibited in these datasets were similar to those in other datasets. At the 0.5 and 0.7 sparsity levels, MF, NLPCA, and UBP did much better than other imputation techniques. Overall, UBP did better than all other algorithms.

this collection was determined before evaluation was performed, and was not modified to emphasize favorable results. To ensure that our results would be applicable for tasks that require generalization, we removed the class labels from each dataset so that only the input features could be used for imputing missing values. We normalized all real-valued attributes to fall within a range from 0 to 1 so that every attribute would carry approximately equal weight in our evaluation. We then removed completely at random[1] $u\%$ of the values from each dataset, where $u \in \{10, 30, 50, 70, 90\}$.

For each dataset, and for each u, we generated 10 datasets with missing values, each using a different random number seed, to make a total of 1550 tasks for evaluation. The task for each of the algorithms was to restore these missing values. We measured error by comparing each predicted (imputed) value with the corresponding original normalized

[1]Other categories of "missingness", besides missing completely at random (MCAR), have been studied [Little and Rubin, 2002], but we restrict our analysis to the imputation of MCAR values.

value, summed over all attributes in the dataset. For nominal (categorical) values, we used Hamming distance, and for real values, we used the squared difference between the original and predicted values. The average error was computed over all of the patterns in each dataset.

We tested 6 imputation algorithms: baseline (BL), fuzzy k-means (FKM), instance-based imputation (IBI), matrix factorization (MF), Nonlinear PCA (NLPCA) and Unsupervised Backpropagation (UBP). With each algorithm-task pair, we tested a variety of algorithmic parameter values. BL has no parameters. With FKM, we varied k (the number of clusters) over the set $\{4, 8, 16\}$, we varied the L_P-norm value for computing distance over the set $\{1, 1.5, 2\}$ (Manhattan distance to Euclidean distance), and the fuzzification factor over the set $\{1.3, 1.5\}$ which were reported to be the most effective values [Li et al., 2004]. With IBI, we used cosine correlation to evaluate similarity, and we varied k (the number of neighbors) over the set $\{1, 5, 21\}$. These values were selected because they were all odd, and spanned the range of intuitively suitable values. With MF, we varied the number of intrinsic values over the set $\{2, 8, 16\}$, and the regularization term over the set $\{0.001, 0.01, 0.1\}$. Again, these values were selected to span the range of intuitively suitable values. With NLPCA, we varied the number of hidden units over the set $\{0, 8, 16\}$, and the number of intrinsic values over the set $\{2, 8, 16\}$. In the case of 0 hidden units, only a single layer of sigmoid units was used. With UBP, the parameters were varied over the same ranges as those of NLPCA. Thus, we imputed missing values in a total of 75950 dataset scenarios. For each algorithm, we found the set of parameters that yielded the best results, and we compared only these best results for each algorithm averaged over the ten runs of differing random seeds.

Figure 8.2 shows two representative comparisons of the error scores obtained by each algorithm at varying levels of sparsity. Comparisons with other datasets generally exhibited similar trends. MF, NLPCA, and UBP did much better than other algorithms when 50% or 70% of the values were missing. No algorithm was best in every case, but UBP achieved the best score in more cases than any other algorithm. Table 8.1 summarizes the results of these comparisons. UBP achieved lower error than the other algorithm in 21 out of 25 pair-wise comparisons, each comparing imputation scores with 31 datasets averaged over 10 runs with different random seeds.

We also performed experiments designed to determine whether the improved imputation accuracy of UBP would lead to better classification accuracy. We compared classification accuracy from 9 learning algorithms from the WEKA [Witten and Frank, 2005] toolkit: C4.5 [Quinlan, 1993], backpropagation, nearest neighbor with generalization [Martin, 1995], locally weighted learning [Atkeson et al., 1997], 5-nearest neighbor, ridor (RIpple DOwn RUle learner) [Gaines and Compton, 1995], RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [Cohen, 1995], and naïve Bayes. The learning algorithms were chosen with the intent of being diverse from one another, where diversity is determined using unsupervised meta-learning [Lee and Giraud-Carrier, 2011]. We evaluated each learning algorithm with 10-fold cross-validation at the previously mentioned sparsity levels using each imputation algorithm with the parameters for each algorithm that resulted in the lowest SSE error score for imputation. The classification accuracies from this experiment are shown in Table 2. The best (highest) scores are in bold. The number of times that UBP achieves a better, equal, and lower accuracy on the data sets is shown in Table 8.3 along with the p-value from the Wilcoxon signed ranks test. At sparsity level 0.1, IBI achieves the highest classification accuracy by a narrow margin. NLPCA and UBP perform similarly and their results are not statistically different. From sparsity level 0.3 to sparsity level 0.7, NLPCA and UBP achieve the highest average classification accuracy for all of the learning algorithms and overall (except for naïve Bayes at the 0.3 sparsity level). With more data missing at the higher sparsity levels, the NLPCA and UBP outperform the other methods because they are better able to extract correlations between the features.

At sparsity levels 0.5 to 0.9, UBP is statistically significantly better than NLPCA for improving classification accuracy. With sparser data, the extra phases for initializing the data and refining the intrinsic variables extract more information from the remaining data. At sparsity level 0.9, however, the baseline method achieves the highest classification accuracy.

The number of times imputation with UBP leads to higher, equal, or lower classification accuracy than other imputation methods in pair-wise comparisons, as well as the Wilcoxon signed-ranks test p-values, are shown in Tables 8.3 and 8.4. Table 8.3 shows the counts and significance at each sparsity level. Table 8.4 shows the same information aggregated over all of the sparsity levels. Imputation with UBP leads to higher classification accuracy with all learning algorithms in the majority of cases. Cases where UBP resulted in better accuracy in the majority of cases are shown in bold. The p-values that indicate statistical significance has been demonstrated are also shown in bold. Overall, imputation with UBP increases the classification accuracy more than with any other algorithm, but especially for the 0.5 to 0.9 sparsity levels. In the cases where another imputation method leads to higher accuracy than UBP, the increase is not significant except for the baseline algorithm at the 0.9 sparsity level.

8.5 Conclusions

We presented a method called Unsupervised Backpropagation (UBP) that trains a multi-layer perceptron to fit to a non-linear manifold sampled by partial observations. We demonstrated that UBP is well-suited for the task of imputing missing values in data. We compared results from UBP with 5 other imputation techniques, including baseline, fuzzy k-means, instance-based imputation, matrix factorization, and Nonlinear PCA, with 31 datasets across a range of parameters for each algorithm. UBP predicts missing values with lower error than any of these other methods in the majority of cases. We also demonstrated that using UBP to impute missing values leads to better classification accuracy than any of the other imputation techniques over all, and at more specific sparsity levels than any other imputation technique.

The reason UBP does better than the closely related Nonlinear PCA is because it utilizes a 3-phase training approach that causes the intrinsic points to be arranged in a globally-representative manner, rather than separated into disjoint clusters. This implies that UBP may also be better-suited for use as a manifold learning technique. Ongoing research seeks to demonstrate the utility of UBP in manifold learning tasks. This potential direction is compelling because, unlike many existing manifold learning algorithms, UBP trains a generalizing model, it can operate on incomplete observations, and it requires no neighbor-finding step. We anticipate that these unique properties may make it well-suited for problems that existing algorithms cannot handle.

Table 8.1: A high-level summary of comparisons between UBP and five other imputation techniques. Results are shown for each of 5 sparsity values. Each row in this table summarizes a comparison between UBP and a competitor algorithm for imputing values in 31 datasets. With each dataset, imputation was performed ten times, each time with different values removed at random. The average accuracy over the ten runs was compared. Only the parameter values that maximized accuracy were used with each algorithm. In 21 out of 25 summary comparisons, UBP did better in the majority of tests. The counts for these cases are shown in bold. In 10 summary comparisons, UBP did better in a sufficient number of cases to demonstrate statistical significance by the Wilcoxon signed ranks significance test ($P < 0.05$). The P-values for these cases are shown in bold.

Sparsity	Algorithm	# of cases UBP did better (out of 31)	P-value
0.1	BL	**25**	**≈ 0**
	FKM	**20**	0.128
	IBI	**17**	0.473
	MF	15	0.550
	NLPCA	**19**	0.066
0.3	BL	**23**	**0.004**
	FKM	**19**	0.168
	IBI	**19**	0.200
	MF	**17**	0.519
	NLPCA	**19**	0.273
0.5	BL	**23**	**0.005**
	FKM	**22**	**0.009**
	IBI	**22**	**0.025**
	MF	**19**	0.473
	NLPCA	**20**	0.184
0.7	BL	**20**	**0.018**
	FKM	**23**	**0.029**
	IBI	**22**	**0.024**
	MF	15	0.632
	NLPCA	**16**	0.314
0.9	BL	13	0.573
	FKM	**24**	**0.008**
	IBI	**22**	**0.040**
	MF	**22**	0.088
	NLPCA	10	0.835

Table 8.2: Classification accuracy with 9 supervised learning algorithms measured using 10-fold cross-validation on data with features that had been partially removed and then imputed using various imputation techniques. In each case, the parameters for the imputation algorithm that resulted in the smallest error were used. The highest classification accuracy in each task is shown in bold. When 10% of the values were imputed, IBI resulted in the best classification accuracy. When 50% or 70% of the values were imputed, UBP resulted in the best classification accuracy. In the largest number of cases, and on average, UBP resulted in the best classification accuracy. In cases where another algorithm did better, the margin was typically small.

		C4.5	NB	BP	NNG	LWL	RID	IB5	RIP	RF	AVE
0.1	BL	0.766	0.721	0.780	0.765	0.683	0.757	0.762	0.755	0.795	0.754
	MF	0.776	0.720	0.798	0.772	0.686	0.762	0.772	0.762	0.802	0.761
	FKM	0.772	0.722	0.784	0.768	0.685	0.758	0.764	0.759	0.794	0.756
	IBI	**0.785**	**0.727**	**0.801**	**0.777**	**0.696**	0.768	**0.773**	**0.774**	**0.804**	**0.767**
	NLPCA	0.779	0.714	0.796	0.771	0.686	**0.772**	0.769	0.763	0.802	0.761
	UBP	0.780	0.718	0.798	0.775	0.686	0.768	0.771	0.762	0.800	0.762
0.3	BL	0.723	0.692	0.731	0.716	0.645	0.701	0.697	0.708	0.744	0.706
	MF	0.728	0.687	0.749	0.722	0.662	0.716	0.712	0.719	0.747	0.716
	FKM	0.718	0.681	0.729	0.711	0.646	0.701	0.697	0.704	0.733	0.702
	IBI	0.726	**0.693**	0.744	0.719	0.664	0.709	0.713	0.712	0.744	0.714
	NLPCA	**0.735**	0.683	**0.745**	**0.724**	**0.662**	0.718	**0.715**	0.721	**0.754**	**0.718**
	UBP	0.730	0.678	0.743	0.723	0.658	**0.726**	0.714	**0.722**	0.746	0.715
0.5	BL	0.677	0.653	0.678	0.667	0.605	0.643	0.640	0.656	0.692	0.657
	MF	0.675	0.650	0.698	0.671	0.620	0.667	0.655	0.660	0.694	0.666
	FKM	0.666	0.637	0.678	0.656	0.610	0.636	0.632	0.647	0.682	0.649
	IBI	0.675	0.652	0.682	0.660	0.617	0.648	0.644	0.649	0.680	0.656
	NLPCA	**0.687**	0.640	0.698	0.672	0.621	0.672	0.658	0.665	0.699	0.668
	UBP	0.686	**0.643**	**0.698**	**0.674**	**0.631**	**0.676**	**0.660**	**0.675**	**0.701**	**0.672**
0.7	BL	**0.623**	0.580	0.624	0.605	0.571	0.597	0.581	0.599	0.628	0.601
	MF	0.618	0.598	0.626	0.594	0.590	0.609	0.583	0.605	0.615	0.604
	FKM	0.609	0.572	0.615	0.593	0.563	0.588	0.576	0.592	0.614	0.591
	IBI	0.607	0.594	0.613	0.599	0.569	0.584	0.579	0.594	0.619	0.595
	NLPCA	0.617	0.581	**0.630**	**0.609**	0.591	0.600	**0.598**	0.604	0.627	0.607
	UBP	**0.623**	**0.586**	0.629	0.607	**0.596**	**0.622**	0.596	**0.614**	**0.633**	**0.612**
0.9	BL	**0.540**	0.457	0.536	0.497	0.526	**0.531**	**0.523**	**0.534**	**0.536**	**0.520**
	MF	0.532	**0.493**	0.531	0.495	0.525	0.523	0.491	0.528	0.509	0.514
	FKM	0.537	0.474	0.533	0.486	0.517	0.521	0.516	0.522	0.540	0.516
	IBI	0.521	0.484	0.521	0.489	0.520	0.523	0.511	0.523	0.520	0.512
	NLPCA	0.533	0.434	**0.542**	0.499	**0.528**	0.514	0.500	0.514	0.508	0.508
	UBP	0.533	**0.493**	0.529	**0.505**	0.521	0.526	0.503	0.528	0.522	0.518
AVE	BL	0.666	0.621	0.670	0.650	0.606	0.646	0.641	0.650	0.679	0.648
	MF	0.666	**0.630**	0.680	0.651	0.617	0.656	0.643	0.655	0.673	0.652
	FKM	0.660	0.617	0.668	0.643	0.604	0.641	0.637	0.645	0.673	0.643
	IBI	0.663	**0.630**	0.672	0.649	0.613	0.646	0.644	0.651	0.673	0.649
	NLPCA	**0.670**	0.610	**0.682**	0.655	**0.618**	0.655	0.648	0.654	0.678	0.652
	UBP	**0.670**	0.624	0.679	**0.657**	**0.618**	**0.664**	**0.649**	**0.660**	**0.680**	**0.656**

Table 8.3: **Counts:** The number of times classification accuracy with UBP-imputed data was better, equal, and worse than classification accuracy with the algorithm specified in the column header. Cases where UBP did better more times than the compared imputation method are bolded. **P-values:** The one-tailed P-value computed using the Wilcoxon Signed-Rank Significance Test. Cases where UBP did significantly better ($P < 0.05$) are bolded.

			C4.5	NB	BP	NNg	LWL	Rid	IB5	RIP	RF	Overall
0.1	Counts	BL	**22,1,8**	**16,4,11**	**24,1,6**	**18,4,9**	**18,4,9**	**21,1,9**	**20,1,10**	**20,0,11**	**19,0,12**	**139,16,62**
		MF	**17,2,12**	**18,1,12**	**17,2,12**	**17,2,12**	**15,7,9**	**18,2,11**	**17,1,13**	**18,3,10**	**17,1,13**	**119,17,81**
		FKM	**21,2,8**	**14,5,12**	**20,1,10**	**22,0,9**	**16,5,10**	**20,3,8**	**21,1,9**	**19,1,11**	**19,2,10**	**134,17,66**
		IBI	13,3,15	12,3,16	15,1,15	**19,2,10**	12,5,14	**15,2,14**	**16,0,15**	15,0,16	12,1,18	**102,16,99**
		NLPCA	**13,6,12**	**12,8,11**	**15,8,8**	**14,7,10**	**13,10,8**	10,6,15	**12,9,10**	**15,8,8**	**10,7,14**	**89,54,74**
	P-values	BL	**0.001**	0.217	**0.001**	**0.035**	0.087	**0.005**	**0.005**	**0.009**	0.088	**≈ 0**
		MF	0.087	0.217	0.225	0.060	0.299	0.065	0.387	0.120	0.475	**0.005**
		FKM	**0.010**	0.315	**0.019**	**0.014**	0.187	**0.027**	**0.018**	**0.024**	**0.046**	**≈ 0**
		IBI	0.616	0.855	0.363	0.182	0.893	0.381	0.346	0.703	0.718	0.783
		NLPCA	0.543	0.228	0.185	0.124	0.093	0.774	0.173	0.093	0.784	0.076
0.3	Counts	BL	**18,1,12**	9,4,18	**19,1,11**	**16,3,12**	**21,2,8**	**19,3,9**	**19,2,10**	**17,1,13**	**12,2,17**	**121,16,80**
		MF	**16,3,12**	**14,3,14**	**11,2,18**	**16,1,14**	**15,4,12**	**21,2,8**	**20,2,9**	**17,1,13**	**16,3,12**	**113,17,87**
		FKM	**19,2,10**	**18,1,12**	**22,0,9**	**19,2,10**	**21,2,8**	**21,1,9**	**23,1,7**	**20,0,11**	**18,0,13**	**143,9,65**
		IBI	**16,1,14**	13,1,17	**18,0,13**	**16,2,13**	12,3,16	**19,2,10**	**17,0,14**	**20,0,11**	**18,0,13**	**111,9,97**
		NLPCA	10,4,17	12,5,14	12,5,14	9,8,14	12,6,13	**18,5,8**	**14,5,12**	**13,6,12**	9,6,16	87,38,92
	P-values	BL	0.236	0.905	**0.029**	0.135	**0.008**	**0.004**	**0.026**	0.157	0.643	**≈ 0**
		MF	0.320	0.734	*0.952*	0.459	0.524	**0.018**	0.109	0.296	0.281	0.167
		FKM	**0.025**	0.217	**0.009**	**0.030**	**0.026**	**0.002**	**0.006**	**0.017**	0.065	**≈ 0**
		IBI	0.255	0.848	0.246	0.182	0.784	**0.008**	0.318	0.085	0.360	**0.036**
		NLPCA	0.852	0.745	0.490	0.790	0.676	**0.049**	0.315	0.637	*0.966*	0.775
0.5	Counts	BL	**16,2,13**	12,4,15	**18,2,11**	**17,2,12**	**19,3,9**	**22,3,6**	**15,3,13**	**18,2,11**	13,2,16	**119,19,79**
		MF	**18,1,12**	13,3,15	**19,1,11**	**19,1,11**	**15,4,12**	**19,3,9**	**17,0,14**	**23,0,8**	**16,2,13**	**120,13,84**
		FKM	**20,1,10**	**16,1,14**	**18,2,11**	**20,3,8**	**18,2,11**	**22,2,7**	**19,2,10**	**20,0,11**	**19,0,12**	**133,13,71**
		IBI	**19,1,11**	**15,2,14**	**22,2,7**	**19,1,11**	**16,4,11**	**24,1,6**	**20,1,10**	**23,1,7**	**24,0,7**	**135,12,70**
		NLPCA	12,7,12	**15,6,10**	**14,4,13**	**19,4,8**	**14,7,10**	**14,5,12**	**14,6,11**	**17,5,9**	13,4,14	**102,39,76**
	P-values	BL	0.176	0.627	**0.044**	0.160	**0.013**	**≈ 0**	0.155	**0.018**	0.381	**≈ 0**
		MF	**0.035**	0.594	0.379	0.311	0.067	**0.022**	0.246	**0.002**	0.101	**≈ 0**
		FKM	**0.010**	0.172	**0.050**	**0.007**	**0.022**	**≈ 0**	**0.014**	**0.003**	0.065	**≈ 0**
		IBI	**0.035**	0.602	**0.011**	**0.012**	0.064	**≈ 0**	**0.019**	**0.001**	**≈ 0**	**≈ 0**
		NLPCA	0.350	0.195	0.590	**0.043**	**0.025**	0.149	0.295	0.055	0.514	**0.004**
0.7	Counts	BL	10,2,19	**17,1,13**	**17,0,14**	**18,1,12**	**19,2,10**	**15,2,14**	**17,1,13**	**16,1,14**	15,0,16	**113,9,95**
		MF	15,1,15	**16,1,14**	**16,1,14**	**19,1,11**	**17,3,11**	**16,2,13**	**18,0,13**	**19,0,12**	**23,0,8**	**117,9,91**
		FKM	15,1,15	**20,0,11**	**16,0,15**	**20,0,11**	**21,1,9**	**17,1,13**	**17,1,13**	**18,0,13**	**18,1,12**	**126,4,87**
		IBI	**18,3,10**	**16,0,15**	**20,1,10**	**21,0,10**	**23,2,6**	**23,2,6**	**17,2,12**	**22,1,8**	**19,1,11**	**138,10,69**
		NLPCA	**14,8,9**	**12,8,11**	**13,6,12**	12,6,13	**11,10,10**	**16,6,9**	**13,6,12**	11,8,12	**14,4,13**	**91,50,76**
	P-values	BL	0.891	0.230	0.240	0.435	**0.025**	0.052	0.116	0.125	0.565	**0.013**
		MF	0.311	0.621	0.242	**0.037**	0.135	0.071	0.063	**0.049**	**0.001**	**≈ 0**
		FKM	0.211	0.088	0.168	0.058	**0.010**	**0.018**	0.121	**0.041**	0.052	**≈ 0**
		IBI	**0.024**	0.654	**0.012**	0.159	**0.002**	**0.002**	0.057	**0.006**	0.064	**≈ 0**
		NLPCA	0.098	0.352	0.277	0.758	0.255	**0.008**	0.511	0.115	0.171	**0.007**
0.9	Counts	BL	8,4,19	**25,0,6**	11,4,16	**19,3,9**	8,4,19	11,3,17	**5,0,26**	11,2,18	8,0,23	87,18,112
		MF	**16,2,13**	13,0,18	**16,1,14**	**19,2,10**	14,2,15	13,2,16	**21,0,10**	**16,3,12**	**22,3,6**	**112,9,96**
		FKM	11,4,16	**17,0,14**	14,2,15	**20,1,10**	**16,2,13**	14,3,14	11,0,20	15,1,15	11,0,20	**103,12,102**
		IBI	**18,3,10**	**20,0,11**	**17,3,11**	**22,2,7**	13,2,16	**17,2,12**	**14,0,17**	**18,0,13**	14,1,16	**121,12,84**
		NLPCA	10,6,15	**18,5,8**	6,5,20	**17,4,10**	10,6,15	**18,5,8**	**14,4,13**	**19,6,6**	**15,3,13**	**93,35,89**
	P-values	BL	*0.939*	**≈ 0**	0.901	0.057	0.841	0.688	*≈ 1*	0.929	*0.997*	*0.974*
		MF	0.466	0.542	0.621	**0.029**	0.667	0.586	0.076	0.281	**0.003**	**0.019**
		FKM	0.835	0.088	0.735	**0.014**	0.357	0.312	*0.972*	0.282	*0.993*	0.581
		IBI	0.057	*0.098*	0.077	**0.003**	0.491	0.287	0.822	0.173	0.557	**0.010**
		NLPCA	0.676	**≈ 0**	*0.997*	0.148	0.732	**0.021**	0.294	**0.001**	0.077	**0.002**

Table 8.4: **Counts:** The number of times that classification accuracy with UBP-imputed data was better, approximately equal (to 4 significant figures), and worse than classification accuracy with the algorithm specified in the column header. Cases where UBP is not better more times than the compared imputation method are bolded. **P-values:** The one-tailed P-value computed using the Wilcoxon Signed-Rank Significance Test. Cases where UBP is significantly better ($P < 0.05$) are bolded.

		C4.5	NB	BP	NNg	LWL	Rid	IB5	RIP	RF	Overall
Counts	BL	**74,10,71**	**79,13,63**	**89,8,58**	**88,13,54**	**85,15,55**	**88,12,55**	**76,7,72**	**82,6,67**	67,4,84	**579,78,428**
	MF	**82,9,64**	**74,8,73**	**79,7,69**	**90,7,58**	**76,20,59**	**87,11,57**	**93,3,59**	**93,7,55**	**94,9,52**	**581,65,439**
	FK	**86,10,59**	**85,7,63**	**90,5,60**	**101,6,48**	**92,12,51**	**94,10,51**	**91,5,59**	**92,2,61**	**85,3,67**	**639,55,391**
	IB	**84,11,60**	**76,6,73**	**92,7,56**	**97,7,51**	**76,16,63**	**98,9,48**	**84,3,68**	**98,2,55**	**87,3,65**	**607,59,419**
	NL	59,31,65	**69,32,54**	60,28,67	**71,29,55**	**60,39,56**	**76,27,52**	**67,30,58**	**75,33,47**	61,24,70	**462,216,407**
P-values	BL	0.215	**0.039**	**0.003**	**0.006**	**0.001**	$\approx$ **0**	0.092	**0.008**	0.758	$\approx$ **0**
	MF	**0.032**	0.636	0.523	**0.004**	0.101	**0.002**	**0.008**	**0.001**	$\approx$ **0**	$\approx$ **0**
	FK	**0.003**	**0.015**	**0.003**	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**	**0.005**	$\approx$ **0**	0.058	$\approx$ **0**
	IB	**0.003**	0.657	**0.001**	$\approx$ **0**	0.084	$\approx$ **0**	0.051	$\approx$ **0**	**0.019**	$\approx$ **0**
	NL	0.454	**0.008**	0.754	0.127	0.121	**0.001**	0.149	**0.001**	0.454	$\approx$ **0**

Chapter 9

Waffles: A Machine Learning Toolkit

Abstract: We present a breadth-oriented collection of cross-platform command-line tools for researchers in machine learning called *Waffles*. The *Waffles* tools are designed to offer a broad spectrum of functionality in a manner that is friendly for scripted automation. All functionality is also available in a C++ class library. *Waffles* is available under the GNU Lesser General Public License.

9.1 Introduction

Although several open source machine learning toolkits already exist [Sonnenburg et al., 2007], many of them implicitly impose requirements regarding how they can be used. For example, some toolkits require a certain platform, language, or virtual machine. Others are designed such that tools can only be connected together with a specific plug-in, filter, or signal/slot architecture. Unfortunately, these interface differences create difficulty for those who have become familiar with a different methodology, and for those who seek to use tools from multiple tookits together. Toolkits that use a graphical interface may be convenient for performing common experiments, but become cumbersome when the user wishes to use a tool in a manner that was not foreseen by the interface designer, or to automate common and repetitive tasks.

Waffles is a collection of tools that seek to provide a wide diversity of useful operations in machine learning and related fields without imposing unnecessary process or interface restrictions on the user. This is done by providing simple command-line interface (CLI) tools that perform basic tasks. The CLI is ideal for this purpose because it is well-established, it is available on most common operating systems, and it is accessible through most common programming languages. Since these tools perform operations at a fairly granular level, they can be used in ways not foreseen by the interface designer.

As an example, consider an experiment involving the following seven steps:

1. Use cross-validation to evaluate the accuracy of a bagging ensemble of one-hundred decision trees for classifying the *lymph* data set (available at `http://MLData.org`).
2. Separate this data set into a matrix of input-features and a matrix of output-labels.
3. Convert input-features to real-valued vectors by representing each nominal attribute as a categorical distribution over possible values.
4. Use principal component analysis to reduce the dimensionality of the feature-vectors.
5. Use cross-validation to evaluate the accuracy of the same model on the data with reduced features.
6. Train the model using all of the reduced-dimensional data.
7. Visualize the model-space represented by the ensemble.

These seven operations can be performed with Waffles tools using the following CLI commands:

1.
   ```
   waffles_learn crossvalidate lymph.arff bag 100 decisiontree end
   ```

2.
   ```
   waffles_transform dropcolumns lymph.arff 18 > features.arff
   waffles_transform dropcolumns lymph.arff 0-17 > labels.arff
   ```

3.
   ```
   waffles_transform nominaltocat features.arff > f_real.arff
   ```

4.
   ```
   waffles_dimred pca f_real.arff 2 > f_reduced.arff
   ```

5.
   ```
   waffles_transform mergehoriz f_reduced.arff labels.arff > all.arff
   waffles_learn crossvalidate all.arff bag 100 decisiontree end
   ```

6.
   ```
   waffles_learn train all.arff bag 100 decisiontree end > ensemble.model
   ```

7. `waffles_plot model ensemble.model all.arff 0 1`

The cross-validation performed in step 1 returns a predictive accuracy score of 0.781. Step 5 returns a predictive accuracy score of 0.705. The plot generated by step 7 is shown in Figure 9.1.

It is certainly conceivable that a graphical interface could be developed that would make it easy to perform an experiment like this one. Such an interface might even provide some mechanism to automatically perform the same experiment over an array of data sets, and using an array of different models. If, however, the user needs to vary a parameter specific to the experiment, such as the number of principal components, or a model-specific parameter, such as the number of trees in the ensemble, the benefits of a graphical interface are quickly overcome by additional complexity. By contrast, a simple script that calls CLI commands to perform machine learning operations can be directly modified to vary any of the parameters. Additionally, the scripting method can incorporate tools from other toolkits, or even custom-developed tools. Because nearly all programming languages can target CLI applications, there are few barriers to adding custom operations. Graphical tools are unlikely to offer such flexibility.

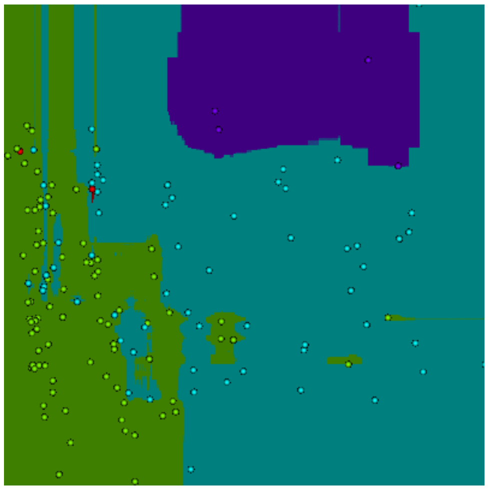
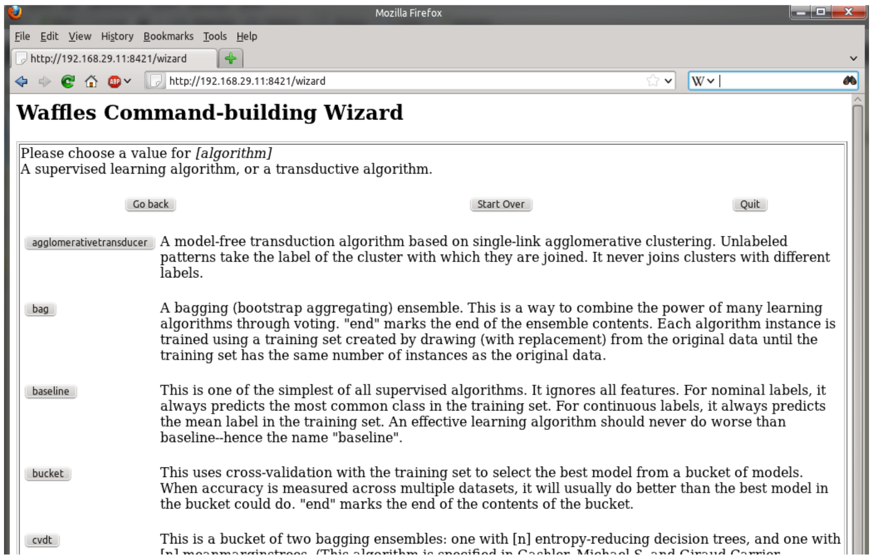


Figure 9.1: The model-space visualization generated by the command in step 7.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.29.11:8421/wizard
http://192.168.29.11:8421/wizard
Go back
Start Over
Quit

Waffles Command-building Wizard

Please choose a value for *[algorithm]*
A supervised learning algorithm, or a transductive algorithm.

agglomerativetransducer	A model-free transduction algorithm based on single-link agglomerative clustering. Unlabeled patterns take the label of the cluster with which they are joined. It never joins clusters with different labels.
bag	A bagging (bootstrap aggregating) ensemble. This is a way to combine the power of many learning algorithms through voting. "end" marks the end of the ensemble contents. Each algorithm instance is trained using a training set created by drawing (with replacement) from the original data until the training set has the same number of instances as the original data.
baseline	This is one of the simplest of all supervised algorithms. It ignores all features. For nominal labels, it always predicts the most common class in the training set. For continuous labels, it always predicts the mean label in the training set. An effective learning algorithm should never do worse than baseline--hence the name "baseline".
bucket	This uses cross-validation with the training set to select the best model from a bucket of models. When accuracy is measured across multiple datasets, it will usually do better than the best model in the bucket could do. "end" marks the end of the contents of the bucket.
cvdt	This is a bucket of two bagging ensembles: one with [n] entropy-reducing decision trees, and one with [n] meanmargintrees. (This algorithm is specified in Gashler, Michael S. and Giraud-Carrier

Figure 9.2: A partial screen shot of the Waffles Wizard tool displayed in a web browser.

9.2 Wizard

One significant reason many people prefer to use tools unified within a graphical interface over scriptable CLI tools is that it can be cumbersome to remember which options are available with CLI tools, and to remember how to construct a syntactically-correct command. We solve this problem by providing a "Wizard" tool that guides the user through a series of forms to construct a command that will perform the desired task. A screen shot of this tool (displayed in a web browser) is shown in Figure 9.2.

Rather than execute the selected operation directly, as most GUI tools do, the Waffles Wizard tool merely displays the CLI command that will perform the operation. The user may paste it directly into a command shell to perform the operation immediately, or the user may choose to incorporate it into a script. This gives the user the benefits of a GUI, without the undesirable tendency to lock the user into an interface that is inflexible for scripted automation.

9.3 Capabilities

In order to hilight the capabilities of Waffles, we compare its functionality with that found in Weka [Hall et al., 2009], which at the time of this writing is the most popular machine learning toolkit by a significant margin. Our intent is not to persuade the reader to choose Waffles instead of Weka, but rather to show that many useful capabilities can be gained by using Waffles in conjunction with Weka, and other toolkits that offer a CLI.

One notable strength of Waffles is in unsupervised algorithms, particularly dimensionality reduction techniques. Waffles tools implement principal component analysis (PCA), isomap [Tenenbaum et al., 2000], locally linear embedding [Roweis and Saul, 2000], manifold sculpting [Gashler et al., 2011], breadth-first unfolding, neuro-PCA, cycle-cut [Gashler and Martinez, 2011a], unsupervised backpropagation and temporal nonlinear dimensionality reduction [Gashler and Martinez, 2011b]. Of these, only PCA is found in Weka. Waffles contains clustering techniques including k-means, k-medoids, agglomerative clustering, and related transduction algorithms including agglomerative transduction, and max-flow/min-cut transduction [Blum and Chawla, 2001].

Waffles provides some of the most-common supervised learning techniques, such as decision trees, multi-layer neural networks, k-nearest neighbor, naive Bayes, and some less-common algorithms, such as Mean-margin trees [Gashler et al., 2008a]. Waffles' collection of supervised algorithms is much smaller than that of Weka, which implements more than 50 classification algorithms. Waffles, however, provides an interface that offers several advantages in many situations. For example, Weka requires the user to set up filters that convert data to types that each algorithm can handle. Waffles automatically handles type conversion when an algorithm receives a type that it is not implicitly designed to handle, while still permitting advanced users to specify custom filters. The Waffles algorithms also implicitly handle multi-dimensional labels. As some algorithm-specific examples, the Waffles implementation of multi-layer perceptron provides the ability to use a diversity of activation functions, and also supplies methods for training recurrent networks. The k-nearest neighbor algorithm automatically supports acceleration structures and sparse training data, so it is suitable for use with problems that require high scalability, such as document classification.

As was demonstrated in the first example in this paper, Waffles features a particularly convenient mechanism for creating bagging ensembles. It also provides a diversity of collaborative filtering algorithms and optimization techniques that are not found in Weka. Waffles also provides tools to perform linear-algebraic operations, and various data-mining tools, including attribute selection and several methods for visualization.

9.4 Architecture

The Waffles tools are organized into several executable applications. These include:

1. **waffles_wizard**, a graphical command-building assistant,
2. **waffles_learn**, a collection of supervised learning techniques and algorithms,
3. **waffles_transform**, a collection of unsupervised data transformations,
4. **waffles_plot**, tools related to visualization,
5. **waffles_dimred**, tools for dimensionality reduction and attribute selection,
6. **waffles_cluster**, tools for clustering data,
7. **waffles_generate**, tools for sampling distributions, manifolds, etc.,
8. **waffles_recommend**, tools related to collaborative filtering, and
9. **waffles_sparse**, tools for learning with sparse matrices.

Each tool contained in each of these applications is implemented as a thin wrapper around functionality in a C++ class library, called *GClasses*. This library is included with Waffles so that any of the functionality available in the Waffles CLI tools can also be linked into C++ applications, or into applications developed in other languages that are capable of linking with C++ libraries. The entire Waffles project is licensed under the GNU Lesser General Public License (LGPL) version 2.1, and also later versions of the LGPL (`http://www.gnu.org/licenses/lgpl.html`). Also, some components are additionally granted more permissive licenses. Waffles uses a minimal set of dependency libraries, and is carefully designed to support cross-platform compatibility. It builds on Linux (with g++), Windows (with Visual C++ Express Edition), OSX (with g++), and most other common platforms. A new version of Waffles has been released approximately every six months since it was first released to the public in 2005. The latest version can be downloaded from `http://waffles.sourceforge.net`. Full documentation for the CLI tools, including many examples, and also documentation for developers seeking to link with the GClasses library can also be found at that site. In order to augment the developer documentation, several demo applications are also included with Waffles, showing how to build machine learning tools that link with functionality in the GClasses library.

Part IV

Conclusions

This part presents the concluding remarks of this dissertation. It begins by presenting a brief summary of the major contributions made by this dissertation, with more technical detail than is given in the introduction of this dissertation. It then identifies some of the remaining challenges in NLDR, and also points out some of the possible research directions that are likely to lead to significant future advances in this field.

Chapter 10

Contributions, and Remaining Challenges

The primary contribution of Chapter 3 is the observation that NLDR can be cast as a graduated optimization problem, which is demonstrated in the Manifold Sculpting algorithm. Casting manifold learning as a graduated optimization problem is significant because graduated optimization provides a mechanism to directly solve the optimization task implicit within NLDR, rather than to seek a solution to an approximation problem, as other NLDR algorithms do. It can be observed that graduated optimization is similar in many ways to an optimization technique called simulated annealing. The differences, however, highlight the importance of this contribution. Simulated annealing does not require the problem to take a particular form, and therefore, is suitable for use with almost all optimization problems. Unfortunately, the computational complexity of simulated annealing increases exponentially with dimensionality, because it essentially uses a Monte Carlo approach to find improvements. By contrast, graduated optimization operates efficiently in high-dimensional spaces, but is only suitable for use with problems that take a particular form. Secondary contributions of this chapter include a method for improving the performance of NLDR with partial supervision.

Chapter 4 contributes a simple and efficient variant of oblique decision tree called Mean Margins Decision Tree. It also makes the observation that adding this alternative tree to decision tree ensembles adds significant homogeneity to the ensemble, to such an extent that very small ensembles containing mean margins trees can outperform much larger Random Forest ensembles.

Chapter 5 contributes a method called CycleCut for pruning shortcut connections from neighborhood graphs based on max-flow/min-cut. This method is superior to the existing approach based on "edge betweenness centrality" because it does not rely on heuristic stopping criteria, it does not remove superfluous edges, and it is additionally suitable for preparing graphs with torroidal topologies for NLDR. Secondary contribution of this chapter include the classification of graph conditions that create difficulty for NLDR techniques, and an efficient method for finding a minimum set of edges to remove in order to ensure that all instances of some structure are removed from a graph.

Chapter 6 contributes a method for neighbor selection called SAFFRON. In contrast with existing approaches, it finds neighborhoods that are more suitable for use with NLDR in more extreme cases. In particular, when combined with CycleCut, it can even produce neighborhoods that are suitable for unfolding self-intersecting manifolds. Secondary contributions of this chapter include a method for computing the dihedral angle between two flats with codimensionality greater than 1, and a method for evaluating the alignment of two flats.

The primary contribution of Chapter 7 is a method called Temporal NLDR for performing NLDR without making the assumption that distances in input observation space are proportional to distances in intrinsic state space. This is significant because this assumption is wrong in many important situations. The removal of this assumption enables NLDR to be used to estimate the state of dynamical systems with accuracy from image-based observations. A secondary contribution of this chapter includes a method for training recurrent neural networks that is less susceptible to problems with local optima than backpropagation-through-time.

Chapter 8 contributes an unsupervised method for training a multi-layer-perceptron called Unsupervised Backpropagation that fits to the manifold represented by data samples. This technique differs from existing approaches for training a multi-layer-perceptron in an unsupervised manner in that it represents the manifold structure only once in the intrinsic space, rather than separate the data into disjoint clusters. A secondary contribution is the demonstration that this method leads to more accurate imputation of missing values in data than existing methods.

Chapter 9 presents an open source toolkit of machine learning algorithms called Waffles. This toolkit offers several algorithms not found in other machine learning toolkits, including all of the algorithms presented in this dissertation.

Although several significant advances were made in the production of this dissertation, many important challenges remain in NLDR. We now attempt to highlight some of the directions that we feel have potential to lead to useful contributions in this field.

One limitation of existing NLDR techniques is that they require enough data samples to thoroughly represent the structure of the implicit underlying manifold. In contrast with these algorithms, humans can often reduce high-dimensional sensory input to a simple understanding of what is intrinsically represented based on only a small number of observations. The relative effectiveness that humans exhibit is probably due in large part to their use of internal models that have been trained through years of experience. In our estimation, the most significant advances in NLDR will be found when they can operate in an incremental manner that makes use of a generalizing internal model of the manifold structure. In order to be effective, such algorithms will likely need to be pre-trained on a large collection of salient observation data so that they can learn to internally represent the sparse codes and assembly hierarchies relevant to the domain, but will then be able to operate effectively with fewer observations.

Another limitation of many NLDR techniques is their reliance on local neighborhood graphs to represent the intrinsic manifold structure. This dissertation presents several techniques to mitigate the difficulties associated with neighborhood graphs, but other problems remain, including the unacceptable computational complexity of finding them in the first place. In contrast with these NLDR approaches, Chapter 8 presented a manifold learning algorithm called Unsupervised Backpropagation (UBP) that performs NLDR without any requirement to find neighbors. Additionally, UBP offers certain other desirable properties, including the ability to operate using sparse observations, and an internal model that facilitates generalization. We conjecture that neural-based approaches for modeling manifold structure will overtake the neighborhood-based approaches. Additionally, as new hardware is becoming increasingly available that facilitates the parallel implementation of neural networks, it is likely that such approaches will soon become much more efficient than neighborhood-based approaches that may not benefit as much from such hardware. We also conjecture that the additional capabilities of neural-based approaches to manifold learning will make them a better choice for general problem solving.

As manifold learning algorithms are used to analyze increasingly complex problems, it will likely become apparent that many problems are not represented well as structured sets of vectors. Such problems will probably require solutions that involve segmenting manifolds into hierarchies, or at least separate parts, and learning such parts independently. An important remaining advance in manifold learning, therefore, will involve techniques for identifying how overlapping parts of separate representations fit together to form a complete whole. This challenge may be exacerbated by the condition that separate parts may not even be represented with the same number of intrinsic variables. Such advances will likely also lead to ensemble techniques in manifold learning that will increase accuracy.

Although much work remains in this field, manifold learning techniques are already beginning to demonstrate capabilities that were previously thought to be exclusive to human brains. As research advances in this field, we are optimistic that our understanding of human intelligence will increase, and that manifold learning will one day be a significant component in machines that exhibit a breadth of capability with a diversity of problems that are currently considered too challenging for machines.

References

E. Acuña and C. Rodriguez. The treatment of missing values and its effect on classifier accuracy. In David Banks, Leanna House, Frederick R. McMorris, Phipps Arabie, and Wolfgang Gaul, editors, *Classification, Clustering, and Data Mining Applications*, volume 0, pages 639–647. Springer Berlin Heidelberg, 2004. ISBN 978-3-642-17103-1.

G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005. ISSN 1041-4347.

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25: 821–837, 1964.

D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, and C. Furlanello. mlpy: Machine Learning Python. *arXiv preprint arXiv:1202.6548*, 2012. `http://mloss.org/software/view/66/`.

A. Asuncion and D. J. Newman. UCI machine learning repository, 2007. URL `http://archive.ics.uci.edu/ml/`.

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, February 1997. ISSN 0269-2821. doi: http://dx.doi.org/10.1023/A:1006559212014. URL `http://dx.doi.org/10.1023/A:1006559212014`.

M. Balasubramanian and E. L. Schwartz. The Isomap algorithm and topological stability. *Science*, 295(5552):7a, January 2002. doi: 10.1126/science.295.5552.7a. URL `http://www.sciencemag.org/content/295/5552/7.short`.

R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. A comparison of decision tree ensemble creation techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):173–180, 2007.

M. Belkin and P. Niyogi. Laplacian Eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th Advances in Neural Information Processing Systems*, pages 585–591, 2001.

R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, Princeton, NJ, USA, 1961.

Y. Bengio and M. Monperrus. Non-local manifold tangent learning. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Proceedings of the 17th Advances in Neural Information Processing Systems*, pages 129–136, Cambridge, MA, 2005. MIT Press.

Y. Bengio, H. Schwenk, J. S. Senécal, F. Morin, and J. L. Gauvain. Neural probabilistic language models. *Innovations in Machine Learning*, pages 137–186, 2006.

M. J. Black. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. In *Proceedings of the International Journal of Computer Vision*, pages 329–342, 1996.

A. Blanco, M. Delgado, and M. C. Pegalajar. A real-coded genetic algorithm for training recurrent neural networks. *Neural Networks*, 14(1):93–105, 2001. ISSN 0893-6080. doi: DOI:10.1016/S0893-6080(00)00081-2.

A. Blum and S. Chawla. Learning from labeled and unlabeled data using Graph Mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 19–26. Morgan Kaufmann Publishers Inc., 2001. ISBN 1558607781.

M. Brand. Charting a manifold. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems, 15*, pages 961–968. MIT Press, Cambridge, MA, 2003.

U. Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001. doi: 10.1080/0022250X.2001.9990249.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. URL `http://citeseer.ist.psu.edu/breiman01random.html`.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

M. Breitenbach and G. Z. Grudic. Clustering through ranking on manifolds. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 73–80, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-180-5. URL `http://doi.acm.org/10.1145/1102351.1102361`.

C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005. URL `http://citeseer.ist.psu.edu/article/brown04diversity.html`.

P. J. Burt. Fast filter transform for image processing. *Computer graphics and image processing*, 16(1):20–51, 1981.

R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 96–103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390169.

C. Y. Chen, J. P. Zhang, and R. Fleischer. Distance approximating dimension reduction of riemannian manifolds. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(1):208 –217, February 2010. ISSN 1083-4419. doi: 10.1109/TSMCB.2009.2025028.

T. C. E. Cheng and M. Y. Kovalyov. An unconstrained optimization problem is NP-hard given an oracle representation of its objective function: a technical note. *Computers & Operations Research*, 29(14):2087 – 2091, 2002. ISSN 0305-0548. doi: DOI:10.1016/S0305-0548(02)00065-5.

D. Coheh and J. Shawe-Taylor. Daugman's Gabor transform as a simple generative back propagation network. *Electronics Letters*, 26(16):1241–1243, 1990.

W. W. Cohen. Fast effective rule induction. In *In Proceedings of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

J. L. Crowley, F. Wallner, and B. Schiele. Position estimation using principal components of range data. *Proceedings of the IEEE International Conference on Robotics and Automation*, 4:3121–3128 vol.4, May 1998. doi: 10.1109/ROBOT.1998.680905.

M. P. Cuéllar, M. Delgado, and M. C. Pegalajar. An application of non-linear programming to train recurrent neural networks in time series prediction problems. *Enterprise Information Systems VII*, pages 95–102, 2006. doi: 10.1007/978-1-4020-5347-4_11.

W. J. Cukierski and D. J. Foran. Using Betweenness Centrality to identify manifold shortcuts. *Data Mining Workshops, International Conference on*, 0:949–958, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/ICDMW.2008.39.

M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, pages 131–156, 1997.

V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Proceedings of the 4th Advances in Neural Information Processing Systems*, pages 705–712, 2002.

P. Demartines and J. Hérault. Curvilinear Component Analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, January 1997. URL `http://citeseer.ist.psu.edu/demartines96curvilinear.html`.

A. Demiriz, K. Bennett, and M. Embrechts. Semi-supervised clustering using genetic algorithms. *Artificial Neural Networks in Engineering*, pages 809–814, 1999. URL `http://citeseer.ist.psu.edu/demiriz99semisupervised.html`.

J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

E. Diday. Recent progress in distance and similarity measures in pattern recognition. In *Second International Joint Conference on Pattern Recognition*, pages 534–539, 1974.

T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000. URL `http://citeseer.ist.psu.edu/dietterich00ensemble.html`.

T. G. Dietterich. Ensemble learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second edition*, pages 405–408. MIT Press, 2002.

D. Donoho and C. Grimes. Hessian Eigenmaps: locally linear embedding techniques for high dimensional data. *National Academy of Sciences of the United States of America*, 100(10):5591–5596, 2003.

A. Farhangfar, L. Kurgan, and J. Dy. Impact of imputation of missing values on classification error for discrete data. *Pattern Recognition*, 41:3692–3705, 2008.

D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *From animals to animats*, 3:421–430, 1994.

L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962. ISBN 978-0-691-14667-6.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the European Conference on Computational Learning Theory*, pages 23–37, 1995. URL `http://citeseer.ist.psu.edu/article/freund95decisiontheoretic.html`.

B. R. Gaines and P. Compton. Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, 5:211–228, November 1995. ISSN 0925-9902. doi: 10.1007/BF00962234. URL `http://dl.acm.org/citation.cfm?id=218246.218250`.

M. S. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, MLOSS 12:2383–2387, July 2011. ISSN 1532-4435. URL `http://www.jmlr.org/papers/volume12/gashler11a/gashler11a.pdf`.

M. S. Gashler and T. Martinez. Tangent space guided intelligent neighbor finding. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2617–2624. IEEE Press, July 2011a.

M. S. Gashler and T. R. Martinez. Temporal nonlinear dimensionality reduction. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1959–1966. IEEE Press, 2011b.

M. S. Gashler and T. R. Martinez. Robust manifold learning with CycleCut. *Connection Science*, 24(1):57–69, 2012. doi: 10.1080/09540091.2012.664122. URL `http://www.tandfonline.com/doi/abs/10.1080/09540091.2012.664122`.

M. S. Gashler, C. Giraud-Carrier, and T. R. Martinez. Decision tree ensemble: Small heterogeneous is better than large homogeneous. In *Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08.*, pages 900–905. IEEE Press, December 2008a. doi: 10.1109/ICMLA.2008.154.

M. S. Gashler, D. Ventura, and T. R. Martinez. Manifold learning by graduated optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, PP(99):1 –13, 2011. ISSN 1083-4419. doi: 10.1109/TSMCB.2011.2151187.

Michael S. Gashler, Dan Ventura, and Tony Martinez. Iterative non-linear dimensionality reduction with Manifold Sculpting. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Proceedings of the 20th Advances in Neural Information Processing Systems*, pages 513–520. MIT Press, Cambridge, MA, 2008b.

X. Geng, D. C. Zhan, and Z. H. Zhou. Supervised nonlinear dimensionality reduction for visualization and classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(6):1098 –1107, December 2005. ISSN 1083-4419. doi: 10.1109/TSMCB.2005.850151.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.58871.

M. Hein and M. Maier. Manifold denoising. In *Proceedings of the 19th Advances in Neural Information Processing Systems*, pages 561–569, Cambridge, MA, 2006. MIT Press.

G. E. Hinton. Generative back-propagation. *Abstracts 1st INNS*, 1988.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 28(313 (5786)):504–507, July 2006.

T. K. Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, Los Alamitos, CA, USA, 1995. IEEE Computer Society. ISBN 0-8186-7128-9. doi: http://doi.ieeecomputersociety.org/10.1109/ICDAR.1995.598994.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

D. Huang, Y. Zhang, and X. R. Pu. Manifold-based learning and synthesis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(3):592 –606, June 2009. ISSN 1083-4419. doi: 10.1109/TSMCB.2008.2007499.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

O. C. Jenkins and M. J. Matarić. A spatio-temporal extension to Isomap nonlinear dimension reduction. In *Proceedings of the 21st International Conference on Machine Learning*, page 56. ACM, 2004.

K. B. Jonathan, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is “nearest neighbor” meaningful? In *Proceedings of the International Conference on Database Theory*, pages 217–235, 1999.

M. P. Jones. Indicator and stratificaion methods for missing explanatory variables in multiple linear regression. *Journal of the American Statistical Association*, 91:222–230, 1996.

N. Keeratipranon, F. Maire, and H. Huang. Manifold learning for robot navigation. *International Journal of Neural Systems*, 16:5:383–392, October 2006.

D. King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009. `http://mloss.org/software/view/83/`.

T. Kohonen. Self-organizing maps. *Series in Information Sciences*, 30, 1997.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. ISSN 0018-9162.

L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles. *Machine Learning*, 51:181–207, 2003. URL `http://citeseer.ist.psu.edu/kuncheva03measures.html`.

S. Lafon. *Diffusion Maps and Geometric Harmonics*. PhD thesis, Yale University, 2004.

K. Lakshminarayan, S. A. Harp, R. Goldman, T. Samad, et al. Imputation of missing data using machine learning techniques. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 140–145, 1996.

M. H. C. Law, N. Zhang, and A. K. Jain. Nonlinear manifold learning for data stream. Technical Report MSU-CSE-03-5, Department of Computer Science and Enginering, Michigan State University, 2004. URL `http://citeseer.ist.psu.edu/691968.html`.

J. Lee and C. Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

J. Lee, A. Lendasse, N. Donckers, and M. Verleysen. A robust non-linear projection method. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 13–20, Bruges (Belgium), 2000.

E. Levina and P. J. Bickel. Maximum likelihood estimation of intrinsic dimension. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 777–784. MIT Press, Cambridge, MA, 2005.

D. Li, J. Deogun, W. Spaulding, and B. Shuart. Towards missing data imputation: A study of fuzzy k-means clustering method. In Shusaku Tsumoto, Roman Slowinski, Jan Komorowski, and Jerzy Grzymala-Busse, editors, *Rough Sets and Current Trends in Computing*, volume 3066 of *Lecture Notes in Computer Science*, pages 573–579. Springer Berlin / Heidelberg, 2004.

Q. Li, Y. P. Chen, and Z. Lin. Filtering Techniques For Selection of Contents and Products. *Personalization of Interactive Multimedia Services: A Research and development Perspective*, 2008.

R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data.* Wiley series in probability and mathematical statistics. Probability and mathematical statistics. Wiley, 2002. ISBN 9780471183860. URL `http://books.google.com/books?id=aYPwAAAAMAAJ`.

J. Luengo. *Soft Computing based learning and Data Analysis: Missing Values and Data Complexity.* PhD thesis, Department of Computer Science and Artificial Intelligence, University of Granada, 2011.

P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Science, Calcutta*, volume 12, page 49, 1936.

B. Martin. Instance-based learning: nearest neighbour with generalisation. Technical Report 95/18, University of Waikato, Department of Computer Science, 1995.

D. Y. Meng, Y. Leung, T. Fung, and Z. B. Xu. Nonlinear dimensionality reduction of data lying on the multicluster manifold. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):1111 –1122, August 2008. ISSN 1083-4419. doi: 10.1109/TSMCB.2008.925663.

K. Menger. Statistical metrics. *Proceedings of the National Academy of Sciences of the United States of America*, 28(12):535, 1942.

M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. In Y. Chauvin and D. Rumelhart, editors, *Backpropagation: Theory, architectures, and applications*, pages 137–169. Hillsdale, NJ: Lawrence Erlbaum Associates, 1995.

S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

S. K. Nayar, S. A. Nene, and H. Murase. Subspace methods for robot vision. *IEEE Transactions on Robotics and Automation*, 12(5):750–758, October 1996. ISSN 1042-296X. doi: 10.1109/70.538979.

S. D. Nikolopoulos and L. Palios. Hole and antihole detection in graphs. In *Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms*, pages 850–859, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. ISBN 0-89871-558-X.

K. Nomizu and H. Ozeki. The existence of complete Riemannian metrics. *Proceedings of the American Mathematical Society*, 12(6):889–891, 1961. ISSN 0002-9939.

D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. URL `http://citeseer.ist.psu.edu/opitz99popular.html`.

A. H. Peterson and T. R. Martinez. Estimating the potential for combining learning models. In *Proceedings of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.

R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6:21–45, 2006.

M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

F. Pourraz and J. L. Crowley. Continuity properties of the appearance manifold for mobile robot position estimation. In *Proceedings of the 2nd IEEE Workshop on Perception for Mobile Agents*, page 2001. IEEE Press, 1998.

J. R. Quinlan. Unknown attribute values in induction. In *Proceedings of the 6th International Machine Learning Workshop*, pages 164–168. Morgan Kaufmann, 1989. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.35.69`.

J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, USA, 1993.

V. V. Raghavan and S. K. M. Wong. A critical analysis of Vector Space Model for information retrieval. *Journal of the American Society for Information Science*, 37(5):279–287, 1986.

A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University, Engineering Department, 1987.

S. Roweis. EM algorithms for PCA and SPCA. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Proceedings of the 10th Advances in Neural Information Processing Systems*. The MIT Press, 1998. URL `http://citeseer.ist.psu.edu/roweis98em.html`.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:9, 1986.

J. W. Sammon. A nonlinear mapping for data structure analysis. In *Proceedings of the IEEE Transactions on Computers*, volume C-18(5), pages 401–409, 1969.

B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295. ACM, 2001. ISBN 1581133480.

J. L. Schafer and J. W. Graham. Missing data: Our view of the state of the art. *Psychological methods*, 7(2):147, 2002. ISSN 1939-1463.

B. Schölkopf, A. J. Smola, and K. R. Müller. Kernel principal component analysis. *Advances in Kernel Methods: Support Vector Learning*, pages 327–352, 1999.

M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig. Non-linear PCA: a missing data approach. *Bioinformatics*, 21(20):3887–3895, 2005.

J. L. Shafer. *Analysis of Incomplete Multivariate Data*, volume 72. Chapman and Hall, London, 1997.

R. N. Shepard and J. D. Carroll. Parametric representation of nonlinear data structures (P. R. Krishnaiah Ed.). In *Proceedings of the 2nd International Symposium on Multivariate Analysis*, pages 561–592. New York: Academic Press, 1965.

J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. Y. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31:1691–1724, 1995.

P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Proceedings of the 8th Advances in Neural Information Processing Systems*, pages 190–196. The MIT Press, 1996. URL `http://citeseer.ist.psu.edu/sollich96learning.html`.

S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K. R. Müller, F. Pereira, C. E. Rasmussen, et al. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. *The Journal of Machine Learning Research*, 11:1799–1802, 2010. `http://mloss.org/software/view/2/`.

E. Sontag. Neural networks for control. *Essays on Control: Perspectives in the Theory and its Applications*, 14:339–380, 1993.

J. P. Spinrad. Finding large holes. *Information Processing Letters*, 39(4):227 – 229, 1991. ISSN 0020-0190. doi: DOI:10.1016/0020-0190(91)90184-J.

S. M. Stigler. Francis Galton's account of the invention of correlation. *Statistical Science*, 4 (2):73–79, 1989. ISSN 0883-4237.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

P. E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1:377–391, 1989.

C. Varini, A. Degenhard, and T. W. Nattkemper. ISOLLE: LLE with geodesic distance. *Neurocomputing*, 69(13-15):1768 – 1771, 2006. ISSN 0925-2312. doi: DOI:10.1016/j.neucom.2005.12.120.

J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19(6):889–899, 2006. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2006.05.014.

P. Vincent and Y. Bengio. Manifold parzen windows. In *Advances in Neural Information Processing Systems 15*, pages 825–832. MIT Press, Cambridge, MA, 2003.

C. Webers, K. Gawande, A. Smola, S. V. N. Vishwanathan, J. Yu, S. Guenter, C. H. Teo, J. McAuley, L. Song, Q. Le, and J. Q. Shi. Elefant, 2009. `http://mloss.org/software/view/38/`.

J. Wei, H. Peng, Y. Lin, Z. Huang, and J. Wang. Adaptive neighborhood selection for manifold learning. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 1, pages 380–384, July 2008. doi: 10.1109/ICMLC.2008.4620435.

K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the 21st International Conference on Machine Learning*, New York, NY, USA, 2004. ACM Press. ISBN 1581138285. doi: http://dx.doi.org/10.1145/1015330.1015345. URL `http://dx.doi.org/10.1145/1015330.1015345`.

K. Q. Weinberger, B. D. Packer, and L. K. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 381–388, 2005.

P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. ISSN 0018-9219.

D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research (JAIR)*, 1:1–34, 1997.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Fransisco, 2nd edition, 2005.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

Z. Wu. The effective energy transformation scheme as a special continuation approach to global optimization with application to molecular conformation. *SIAM Journal on Optimization*, 6:748, 1996.

Z. Zhang and J. Wang. MLLE: Modified locally linear embedding using multiple weights. *Advances in Neural Information Processing Systems*, 19:1593, 2007. ISSN 1049-5258.

Z. Zhang and H. Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing*, 26:313–338, 2002.

Z. Zhang and H. Zha. A domain decomposition method for fast manifold learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *Proceedings of the 16th Advances in Neural Information Processing Systems*. MIT Press, 2004.

T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. Modular toolkit for data processing. *Frontiers in Neuroinformatics*, 2:8, 2009. doi: 10.3389/neuro.11.008.2008. `http://mloss.org/software/view/60/`.