Dept. of Biomedical Engineering and Computational Science

Studies on dimension reduction and feature spaces

Eli Parviainen

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 94/2011

Studies on dimension reduction and feature spaces

Eli Parviainen

Doctoral dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the School of Science for public examination and debate in Auditorium F239a at the Aalto University School of Science (Espoo, Finland) on the 11th of November 2011 at 12 noon.

Aalto University
School of Science
Dept. of Biomedical Engineering and Computational Science

Supervisor
Prof. Jouko Lampinen

Preliminary examiners
Prof. José Alfredo F. Costa, Universidade Federal do Rio Grande do Norte, Brazil
Dr. Marc Strickert, Universität Siegen, Germany

Opponent
Prof. Michel Verleysen, Université Catholique de Louvain, Belgium

Aalto University publication series
DOCTORAL DISSERTATIONS 94/2011

ISBN 978-952-60-4312-8 (pdf)
ISBN 978-952-60-4311-1 (printed)
ISSN-L 1799-4934
ISSN 1799-4942 (pdf)
ISSN 1799-4934 (printed)

Unigrafia Oy
Helsinki 2011

Finland

The dissertation can be read at http://lib.tkk.fi/Diss/

Author
Eli Parviainen

Name of the doctoral dissertation
Studies on dimension reduction and feature spaces

Publisher School of Science

Unit BECS - Department of Biomedical Engineering and Computational Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 94/2011

Field of research Computational Science

Manuscript submitted 9 April 2011 **Manuscript revised** 12 September 2011

Date of the defence 11 November 2011 **Language** English

☒ **Monograph** ☐ **Article dissertation (summary + original articles)**

Abstract

Today's world produces and stores huge amounts of data, which calls for methods that can tackle both growing sizes and growing dimensionalities of data sets. Dimension reduction aims at answering the challenges posed by the latter.

Many dimension reduction methods consist of a metric transformation part followed by optimization of a cost function. Several classes of cost functions have been developed and studied, while metrics have received less attention. We promote the view that metrics should be lifted to a more independent role in dimension reduction research. The subject of this work is the interaction of metrics with dimension reduction. The work is built on a series of studies on current topics in dimension reduction and neural network research. Neural networks are used both as a tool and as a target for dimension reduction.

When the results of modeling or clustering are represented as a metric, they can be studied using dimension reduction, or they can be used to introduce new properties into a dimension reduction method. We give two examples of such use: visualizing results of hierarchical clustering, and creating supervised variants of existing dimension reduction methods by using a metric that is built on the feature space of a neural network. Combining clustering with dimension reduction results in a novel way for creating space-efficient visualizations, that tell both about hierarchical structure and about distances of clusters.

We study feature spaces used in a recently developed neural network architecture called extreme learning machine. We give a novel interpretation for such neural networks, and recognize the need to parameterize extreme learning machines with the variance of network weights. This has practical implications for use of extreme learning machines, since the current practice emphasizes the role of hidden units and ignores the variance.

A current trend in the research of deep neural networks is to use cost functions from dimension reduction methods to train the network for supervised dimension reduction. We show that equally good results can be obtained by training a bottlenecked neural network for classification or regression, which is faster than using a dimension reduction cost.

We demonstrate that, contrary to the current belief, using sparse distance matrices for creating fast dimension reduction methods is feasible, if a proper balance between short-distance and long-distance entries in the sparse matrix is maintained. This observation opens up a promising research direction, with possibility to use modern dimension reduction methods on much larger data sets than which are manageable today.

Keywords dimension reduction, neural network, metric, visualization

ISBN (printed) 978-952-60-4311-1 **ISBN (pdf)** 978-952-60-4312-8

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Espoo **Location of printing** Helsinki **Year** 2011

Pages 107 **The dissertation can be read at** http://lib.tkk.fi/Diss/

Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä
Eli Parviainen

Väitöskirjan nimi
Ulotteisuudenpienennyksestä ja piirreavaruuksista

Julkaisija Perustieteiden korkeakoulu

Yksikkö Lääketieteellisen tekniikan ja laskennallisen tieteen laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 94/2011

Tutkimusala Laskennallinen tekniikka

Käsikirjoituksen pvm 09.04.2011 **Korjatun käsikirjoituksen pvm** 12.09.2011

Väitöspäivä 11.11.2011 **Kieli** Englanti

☒ **Monografia** ☐ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Nykypäivänä tuotetaan ja tallennetaan valtavia määriä tietoa, joten menetelmien pitäisi selviytyä sekä kasvavasta datamäärästä että datan kasvavasta ulotteisuudesta. Ulotteisuudenpienennysmenetelmät pyrkivät vastaamaan jälkimmäiseen haasteeseen.

Monet ulotteisuudenpienennysmenetelmät koostuvat metrisestä muunnoksesta, jota seuraa kustannusfunktion optimointi. Useita erityyppisiä kustannusfunktioita on kehitetty ja tutkittu, mutta metriikat ovat jääneet vähemmälle huomiolle. Tämän työn näkökulma ulotteisuudenpienennykseen on nostaa metriikat itsenäisempään rooliin. Työn aihe on metriikan ja ulotteisuudenpienennyksen vuorovaikutus. Työ rakentuu joukosta ulotteisuudenpienennyksen ja neuroverkkotutkimuksen ajankohtaisiin kysymyksiin liittyviä tutkielmia. Neuroverkkoja käytetään sekä ulotteisuudenpienennyksen työvälineenä että tutkimuskohteena.

Esittämällä mallinnuksen tai klusteroinnin tulokset metriikan muodossa voidaan niitä tutkia ulotteisuudenpienennysmenetelmillä, tai käyttää niitä tuomaan uusia ominaisuuksia ulotteisuudenpienennykseen. Kaksi esimerkkiä ovat hierarkkisen klusteroinnin tulosten visualisointi, ja ohjattujen ulotteisuudenpienennysmenetelmien luominen neuroverkkomallin piirreavaruuteen pohjautuvan metriikan avulla. Klusteroinnin ja ulotteisuudenpienennyksen yhdistäminen tuottaa uuden tavan luoda tilatehokkaita visualisointeja, joissa yhdistetään hierarkkinen rakenne tietoon klustereiden etäisyyksistä.

Työssä tutkitaan äskettäin kehitetyn extreme learning machine -nimisen neuroverkkomenetelmän piirreavaruutta. Tällaiselle neuroverkolle esitetään uusi tulkinta, ja todetaan tarve parametrisoida menetelmä painojen varianssia käyttäen. Tällä on käytännön merkitystä extreme learning machine -menetelmän käyttöön, sillä nykykäytäntö korostaa piiloyksikkömäärän tärkeyttä ja jättää painojen varianssin huomiotta.

Eräs syvien neuroverkkojen tutkimuksen nykytrendi on käyttää ohjattujen ulotteisuudenpienennysmenetelmien kustannusfunktioita neuroverkon opetukseen. Tässä työssä näytetään, että yhtä hyviin tuloksiin voidaan päästä yksinkertaisesti opettamalla pullonkaulakerroksella varustettu neuroverkko toteuttamaan luokittelu tai regressio. Tämä on nopeampaa kuin ulotteisuudenpienennyskustannuksen käyttö.

Työssä osoitetaan, että toisin kuin nykyään ajatellaan, nopeiden ulotteisuudenpienennysmenetelmien luominen harvaa etäisyysmatriisia käyttäen on toimiva ajatus, mikäli matriisissa säilyy sopiva tasapaino lyhyiden ja pitkien etäisyyksien välillä. Tämä havainto avaa lupaavan suunnan jatkotutkimukselle, ja saattaa mahdollistaa nykyaikaisten ulotteisuudenpienennysmenetelmien käytön nykyistä paljon suuremmilla datoilla.

Avainsanat ulotteisuudenpienennys, neuroverkko, metriikka, visualisointi

ISBN (painettu) 978-952-60-4311-1 **ISBN (pdf)** 978-952-60-4312-8

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Espoo **Painopaikka** Helsinki **Vuosi** 2011

Sivumäärä 107 **Luettavissa verkossa osoitteessa** http://lib.tkk.fi/Diss/

Preface

Work for this thesis was done at Department of Biomedical Engineering and Computational Science (BECS) during 2009-2010, with some finishing touches in spring 2011. Most of funding came from Finnish Funding Agency for Technology and Innovation (TEKES) through a project where I worked before my thesis. In addition to funding, the project provided a chance to take the necessary courses and acquire some experience on machine learning, which enabled rapid completion of the thesis work afterwards. Faculty of Information and Natural Sciences kindly supported finalization of the thesis with some extra money.

I warmly thank my supervisor prof. Jouko Lampinen for allowing me to jump on his family's trampoline, a most enjoyable experience which outshines any recreation I could have devised in an office. I also thank prof. Lampinen for possibility to do my PhD at BECS. In pleasantness thesis work cannot be compared to trampoline jumping, but all the same it provided welcome escapism and occasionally a chance to sprain some braincells.

Not all PhD stories are success stories, and not all tunnels have a light in the end. Towards the end I came to feel so much like an outcast that I shied away from my luckier colleagues. Anyway, the few good memories I have from my time at BECS are of moments shared with fellow students. First of all I thank the poor souls who had to share an office with me: Markus Siivola, who had the good sense to escape to Canada; Juho Kettunen, a constant source of support with his grown-up and mature attitude (he seldom said anything suitable for ears of minors); and especially Jaakko Riihimäki, whose friendliness and patience kept me afloat innumerable times. Jaakko is the only person ever to fool me into eating pinecones, and to force me to see a dentist (these two events are unrelated). I also thank the rest of the gang, in no particular order: Jarno Vanhatalo (awesomely efficient), Pasi Jylänki (awesomely smart), Janne Ojanen (awesomely violent), Jouni Hartikainen (the other QPO; apologies for the Finnish pun), and others.

To Tommi and Aleph I would like to point out, that although my thesis lacks an allusion to Superpenguin (admittedly a grave fault in any scientific work), everything in my thesis has much lower computational complexity than anything in Tommi's thesis. Yes, some of us do consider it a virtue.

Finally, my gratitude goes to my parents, whose influence I recognize in my thinking even today. My mother ingrained the value of knowledge by insisting, that the only valid reson to quit the table before the meal is over is to look things up in a dictionary. My father taught me that often the best way to improve a design is to leave things out.

Espoo, September 12, 2011,

Eli Parviainen

Contents

Preface **1**

Contents **3**

1 Introduction **7**
1.1 Outline of the work 8

2 Background **11**
2.1 Basic concepts 11
2.2 A taxonomy of dimension reduction methods 19
2.3 Metrics 20
2.4 Cost functions 22
2.5 Supervised dimension reduction 27

3 Metrics bridging models and dimension reduction **35**
3.1 Metrics that describe results of modeling 35
3.2 Visualizing hierarchical clustering 38
3.3 Comparison of feature space and LM visualizations 40
3.4 Conclusions 42

4 Feature space of the Extreme Learning Machine **45**
4.1 Random and infinite neural networks 46
4.2 Comparison of feature spaces 47
4.3 Extreme learning machine approximates the neural network kernel 49
4.4 Variance parameter in ELM 52
4.5 On properties of ELM 54
4.6 Conclusions 55

5 Supervised dimension reduction with bottleneck networks **57**
5.1 Feedforward neural networks in dimension reduction 58
5.2 Supervised bottleneck networks 60
5.3 Bottleneck classifier experiments 61
5.4 Bottleneck regression network experiments 64
5.5 Conclusions 70

6 Dimension reduction with a sparse metric **73**
6.1 Existing sparse approaches 73
6.2 Local and global in sparse distance matrices 75
6.3 Conclusions 78

7 Discussion **83**

A Data and parameters **85**
A.1 Description of data sets 85
A.2 Data and parameters of experiments 86

Bibliography **89**

List of abbreviations

AE	autoencoder
BC	bottleneck classifier
BRN	bottleneck regression network
CCA	curvilinear component analysis
CDA	curvilinear distance analysis
COIR	covariance operator inverse regression
DBN	deep belief network
DC	discriminative clustering
DR	dimension reduction
DRR	dimension reduction for regression
ELM	extreme learning machine
GPLVM	Gaussian process latent variable model
HC	hierarchical clustering
HLLE	Hessian locally linear embedding
KDR	kernel dimension reduction
KL	Kullback-Leibler (divergence)
kNN	k nearest neighbors
KPCA	kernel principal component analysis
LDA	linear discriminant analysis
LE	Laplacian eigenmap
LMNN	large margin nearest neighbor
LLE	locally linear embedding
MDS	multidimensional scaling
mKDR	manifold kernel dimension reduction
MLP	multilayer perceptron network
MVE	minimum volume embedding
MVU	maximum variance unfolding
NCA	neighborhood component analysis
NNK	neural network kernel
PCA	principal component analysis
PSD	positive semidefinite
RBM	restricted Boltzmann machine
RMS	root mean square (error)
SIR	sliced inverse regression
SNE	stochastic neighbor embedding
SOM	self organizing map
SVM	support vector machine
t-SNE	t-distributed stochastic neighbor embedding

1. Introduction

This work is a monograph built on a series of studies on current topics in dimension reduction and neural network research. What unifies these two subjects is the importance of a metric transformation as a building block. Dimension reduction attempts to transform points of a high-dimensional space into a low-dimensional space, obeying constraints specified by a cost function. Hidden layers of a neural network implement successive transformations that turn data into features. Usually features are found so that they minimize a certain cost. In this work neural networks are used both as a tool and a target for dimension reduction.

Dimension reduction is a topical research subject. Today's world produces and stores huge amounts of data, which calls for methods that can tackle both growing sizes and growing dimensionalities of data sets. Dimension reduction tries to answer the challenges posed by the latter.

Term "dimension reduction" collects under its umbrella several methods with partly overlapping, partly differing uses. Some methods are good for data preprocessing, turning a high-dimensional data into lower-dimensional form for algorithms to use. Others concentrate on recognizing shapes of data clouds, effectively learning data-internal coordinates, and such coordinates can then be used in further processing. Yet others aim to pack data so that the results can be used for predictions like full data, but take less space and are faster to handle. Finally, some methods are developed with visualization in mind, and also results from other methods are often applied in visualizations. The emphasis of this work is on two-dimensional visualizations.

A problem we see in current dimension reduction research is that too often methods are presented as monolithic, even though many consist of a fairly independent metric transformation part followed by optimization of a cost function. This leads to an unmanageable number of seemingly new methods, many of which could be better categorized as combinations of a basic cost function and a new metric, or vice versa. Modularity of presentation would facilitate both development and analysis of dimension reduction methods.

There are several successful cost functions for dimension reduction, based on different ideas of what should be preserved in the low-dimensional representation of data. The role of metrics, on the contrary, has received less attention. We promote the view that metrics should be lifted to a more independent role in dimension reduction research. Shifting the focus on metrics and the interaction of the metric and the cost function opens up several routes for methodological development. It is not necessary to pack all functionality of a dimension reduction method into one cost function. By optimizing the metric separately (e.g. to facilitate clas-

sification) new properties can be easily introduced into existing methods. Dimension reduction can be applied to higher-level entities than raw data by describing such entities (e.g. a clustering result) in the form of a metric. Connections to feature extraction become more clear – a cost function of a dimension reduction method can e.g. guide the feature learning in a neural network, instead of just finding low-dimensional points. And, speeding up dimension reduction can be done by using a sparse metric, instead of using complicated heuristics for choosing points or distances in the optimization algorithm.

This work contributes to surveying the territory between metrics and cost functions. We study the interaction of metrics and dimension reduction from two directions. Two chapters make the dimension reduction method the object of study, and modify a metric to affect behavior of dimension reduction: Chapter 3 uses a metric to convey modeling results into a dimension reduction method, and Chapter 6 develops a dimension reduction method based on a sparse metric. Two chapters take the opposite view, and use dimension reduction to create or understand a metric: in Chapter 4, the feature space of a neural network model is studied, using tools of dimension reduction and analysis of similarity matrices, and Chapter 5 concentrates on guiding feature learning of a neural network by different cost functions.

1.1 Outline of the work

The background in Chapter 2 is a review of common concepts and methods of dimension reduction.

Chapter 3 extends and refines work from [135] , [132], and in a small part, [136]. The key idea of this chapter is to use a metric as a bridge between a model and a dimension reduction method. Two realizations of this idea a studied. First, we visualize results of hierarchical clustering using dimension reduction. Second, we implement supervised variants of existing unsupervised dimension reduction methods by using a metric based on feature space of a supervised neural network model, and present a comparison to a related method.

In Chapter 4 we study extreme learning machines (ELM), a recently developed neural network architecture based on random projections. Main contributions of this chapter are a novel interpretation of ELM, which points out its resemblance to kernel classifiers, and recognition of importance of weight variance as a parameter in ELM training. We also discuss some claims made about ELM, which we feel may have been too strongly presented in earlier work. Most of the study has been published in [134][1], visualizations of feature spaces being new. Only the author's material from the jointly written [134] was chosen in this work.

In Chapter 5 we study bottleneck classifiers, which complement the current research on deep networks for dimension reduction. We show that supervised bottleneck classifiers are a competitive alternative to currently used deep networks for supervised dimension reduction. Evaluation of the cost function of a bottleneck classifier scales linearly in the number of data points, while state-of-the-art methods use cost functions that scale quadratically or worse. Although classification and regression are related tasks from a neural network point of view, super-

[1] in proceedings of International Conference on Knowledge Discovery and Information Retrieval 2010; reused with permission

vised dimension reduction is clearly divided into classification-oriented and regression-oriented research branches. We therefore experiment separately with bottlenecked regression networks, and show that they perform as well as the state-of-the art methods used in dimension reduction for regression. We are not aware of earlier use of neural networks in the field of dimension reduction for regression. Bottleneck classifiers have been studied in [130][2], and part of experiments on bottleneck regression networks are from [131][2].

Chapter 6 presents work on sparse dimension reduction, published in [133]. We demonstrate that, contrary to the current belief, using sparse distance matrices for creating fast dimension reduction methods is a promising idea. We learn that a proper balance between short-distance and long-distance entries in the sparse matrix is needed for good results. The chapter opens a promising direction for further research, and we also discuss some lines for further development.

[2]reused with kind permission from Springer Science+Business Media

2. Background

In this section, we first introduce some background concepts, and describe commonly used dimension reduction methods.

The number of papers published about dimension reduction currently seems to grow exponentially (Fig. 2.1), and it would be impossible to attempt a full account here. We briefly recount some methods that illustrate common principles in how dimension reduction (DR) methods work. Many of the methods presented here have introduced novel ideas to the field, and thereby serve as basic methods which have later been extended or modified. Further reviews on dimension reduction include [24, 54, 23, 208, 109, 183].

2.1 Basic concepts

2.1.1 Dimension reduction in machine learning

Some of most basic problems of machine learning are classification, clustering, density estimation and dimension reduction. Using such a division satisfies humans' need to create clear-cut taxonomies, and is also motivated by the often widely differing algorithms for solving the basic problems. Solutions to all these problems, however, build on the thought that a data point has a different or stronger relationship to a nearby-located point than to a point further away. It is therefore not surprising that sometimes the different problems can be solved by similar methods, and sometimes the borderlines between the problems are not clear-cut.

Many, if not most, modern dimension reduction methods use some kind of density estimate. It is seldom explicitly called a density estimate, but forming graphs with links to nearest neighbors or placing kernels on the data points has the effect of deciding which areas are local, and how strongly local information should be emphasized. This is not very different from using Parzen windows or k-nearest neighbor density estimates [45]. The density estimate interpretation has been exploited in [139], where a DR method of quadratic complexity is turned into a linearly scaling method by replacing a non-parametric Parzen estimate by a Gaussian mixture model.

Dimension reduction is also connected with clustering. Some methods can be interpreted as clustering methods as well as DR methods [95, 43]. Both DR and clustering can be based on similar density estimates and graph-based representations built thereon [8, 194]. Dimension reduction of clustered data has been specifically considered [153, 159], and clustering has been used as a preprocessing step in order to obtain labels for

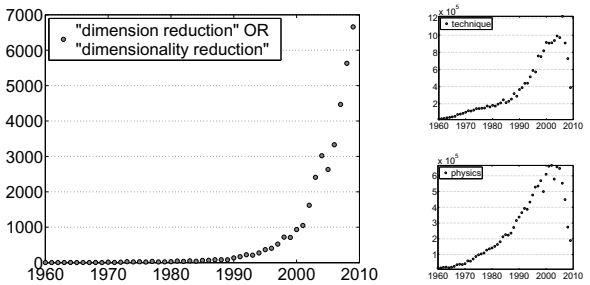


Figure 2.1. 50 years of dimension reduction in Google Scholar. The number of search hits for "dimension reduction" has increased exponentially from 1960s to today, with 6660 hits for 2009 (search results of June 16, 2010). The growth is not explained just by better availability of newer publications, as attested by two searches with more general search terms.

supervised DR [144]. Dimension reduction methods can also be combined with a clustering-based metric to visualize clustering results [132].

Classification, as well as clustering, can often be improved by choosing or learning a metric on data as a preprocessing step [179, 62, 210, 202, 196]. Supervised metrics provided by such methods can be used with dimension reduction algorithms, or they can directly provide a reduced representation, if the output dimensionality of the method can be controlled. In a classification problem, the metric should improve accuracy; in a dimension reduction problem, the metric should lower dimension while worsening accuracy as little as possible. Although the goals differ, technically, the same cost functions and the same algorithms can often be used in metric learning for both tasks.

Performing dimension reduction and learning a data-based metric are thus closely related tasks. In some cases the same DR problem can be formulated either as first learning a metric and then using e.g. its highest principal components as a reduced representation, or as directly learning a reduced-rank metric. Therefore, the two terms are sometimes interchanged. As a metric is often specified as a (positive semidefinite) matrix, also term kernel learning is sometimes used for the metric learning task.

2.1.2 Intrinsic dimensionality

The number of attributes is usually different from dimensionality. In problems with less data points than dimensions, the number of data points dictates the maximum dimensionality of the subspace that the data vectors can span. Sometimes data can contain directions that are mostly noise, or it can have attributes that are mostly irrelevant for the task in hand and can be left out by variable selection. And, even more importantly, data may contain dependencies which effectively lower its dimensionality. The smaller dimensionality caused by dependencies is referred to as the intrinsic dimensionality of data, or the number of degrees of freedom that is sufficient for describing the data.

Determining the intrinsic dimensionality is important when the goal is to describe data concisely but without losing information. In the visualization applications the presentations are forced to 2D or 3D, with the price of greater information loss. In such context the intrinsic dimensionality is often ignored. In this work we mainly concentrate on dimension

reduction as a visualization tool, aiming at 2D results. As the output dimensionality is fixed, we do not explicitly try to estimate the intrinsic dimensionalities.

2.1.3 Curse of dimensionality

Algorithms designed with intuitions based on two- or three-dimensional spaces may face unexpected problems when dealing with high-dimensional data [109, 90, 1]. The phenomenon maybe most relevant for dimension reduction is the concentration of distances. When number of dimension components grows, contribution of any one axis becomes more and more meaningless. It becomes difficult to tell two points apart unless they differ in most dimensions. Most points are roughly equally far from each other, and the distribution of pairwise distances becomes peaked.

An important practical consequence of this is that the concept of nearest neighbor is weakened [15, 77]. When the difference in distances to the nearest and the furthest neighbors becomes less than the measurement noise, algorithms relying on neighborhoods break down. This and other peculiarities of high-dimensional spaces are referred to as the curse of dimensionality.

DiMenSionAlity effects can severely affect distance-based algorithms, and become noticeable already in 5–10 dimensions [109, 55], which is much less than the number of attributes in many modern data sets. That such algorithms can be used at all depends on the basic assumption that the intrinsic dimensionality of the data is fairly low [112].

Nonlinear dimensionality reduction, sometimes thought as a remedy for the curse of dimensionality, implicitly relies on the assumption of a low intrinsic dimensionality. Most DR methods use nearest neighbors or comparisons of pairwise distances, and are thus equally sensitive to the curse of dimensionality as other machine learning methods.

2.1.4 Unfolding manifolds

Some data clouds are curved in shape, and better described with Riemannian than Euclidean coordinates. Data is thought to be a sample from a manifold, which is embedded in a higher-dimensional space. Manifold generalizes the linear subspace by introducing curvilinear coordinates. In small enough scale, manifolds are approximately linear, in same fashion as the surface of the earth appears flat in local view.

The aim of manifold methods is to uncover the intrinsic coordinate system of the manifold from which the data is sampled. Dimension reduction is done using these coordinates instead of the Euclidean coordinates of the embedding space. The manifold coordinates can be approximated by studying the data in a local scale, and then combining local, linear views into a global nonlinear shape. Local views are obtained by restricting the analysis to the nearest neighbors of a data point. There are several ways to work with local neighborhoods. Some of them will be introduced below.

Sometimes term "manifold learning" is used synonymously with nonlinear dimension reduction [109] to emphasize the fact that the methods assume existence of a low-dimensional structure and try to find it, instead of transforming truly high-dimensional data into a low-dimensional form. We continue to speak about nonlinear dimension reduction, being aware of that the main uses of the methods may lie elsewhere than in fighting the curse of dimensionality.

Approximating the manifold coordinates is deservedly one of basic ap-

proaches to non-linear dimension reduction, but should not be seen as a panacea. Complicated manifolds are present in toy problems like the famous Swiss Roll, and systematically collected data (like the Yale and Head Pose data sets we will use later, see Appendix A) can exhibit very clear manifold structure. Data sets collected under more varying conditions are seldom cooperative enough to resemble uniform, dense samples from a manifold. Clustered and non-uniformly sampled data are obvious challenges for methods, that basically assume a single manifold shape well represented by the data cloud. Many manifold methods are reviewed in [183], and are not found to do better than global methods in real problems.

Manifolds are not the only way to look at the structure of data. Alternatives include clustering the data (e.g. self-organizing maps [95] and generative topographic mapping [17]), or emphasizing its natural density structure (e.g. stochastic neighbor embedding [78]). Some methods are able to unfold manifolds, even if they do not explicitly form manifold coordinates (e.g. curvilinear components analysis, [76]). The focus of DR development is still on methods that rely on neighborhoods, but neighborhoods can be exploited without bringing in strong manifold assumptions.

2.1.5 Similarity representation of data

The location of a data point in space can be given by listing its coordinates, one per dimension. This is the representation used in vectorial data. An alternative representation, similarity data, gets rid of the notion of coordinates, and only lists pairwise comparisons between the points. These can be measurements of either similarity or dissimilarity of points.

Similarity representation can handle more diverse data types than vectorial data, e.g. graphs [96], since developing a similarity measure may be easier than converting a complex data object into a vector. For data sets with a small number of very high-dimensional points, like gene data sets with a few thousand persons but hundreds of thousands of measurements, similarity matrices take less space than the data itself, and thus be faster to handle. These properties of similarities have been exploited e.g. in visualizing text documents [35] and gene data [88] .

Pairwise similarity or dissimilarity is the form of data used by most dimension reduction methods. Also some classification and clustering algorithms can be implemented directly on similarities [29, 138]. Some algorithms originally designed for vector data have been directly applied using a list of pairwise similarities as a feature vector [138].

General similarity A general similarity can be specified by a pairwise matrix, with no restrictions on matrix elements (except perhaps the requirement that a point should be maximally similar to itself). Even symmetry is not a necessary requirement: if the similarity describes e.g. closeness of persons, as reported by themselves, two persons may obviously have different views on their relationship.

Similarity and dissimilarity matrices can be converted into each other, although no single "correct" transformation exists. A common transformation, used e.g. in `mdscale` function of Matlab, which can be used if similarities are first normalized into [0,1]-range is

$$d_{ij} = \sqrt{1 - s_{ij}} \ . \tag{2.1}$$

Generally there is no one-to-one correspondence between a similarity matrix and points of a metric space, as a general similarity might specify constraints which do not necessarily yield a meaningful topology [87]. General similarity matrices are encountered in applications where data is originally collected as pairwise comparisons. Such similarities can be embedded in Euclidean spaces only approximately. Slight deviations from metricity are not a problem in DR, since dimension reduction by its nature looks for compromises. Even if the original(dis)similarities stem from a metric high-dimensional space, even the best possible representation in a low-dimensional space cannot usually satisfy all constraints based on high-dimensional point locations.

Positive semidefinite similarity In dimension reduction context one often deals with a restricted class of similarity matrices, the positive semidefinite (PSD) ones. PSD matrices arise when vectorial data is converted into similarity data, or when a metric is learned and constrained to be PSD.

A positive semidefinite matrix $\mathbf{A}$ is a similarity matrix, which fulfills the condition $\mathbf{v}^T\mathbf{A}\mathbf{v} \geq 0$ for all non-zero (real) vectors $\mathbf{v}$. A PSD matrix is symmetric and has positive, real eigenvalues. Matrices of this family correspond to covariance matrices in some, possibly high-dimensional Euclidean space. Corresponding vectorial representation can be recovered by eigendecomposition.

As covariance $s_{ij} = \mathbf{x}_i^T\mathbf{x}_j$ and Euclidean distance d_{ij} are connected through

$$d_{ij}^2 = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{x}_j + \mathbf{x}_j^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j, \tag{2.2}$$

PSD matrices are naturally converted into squared distance matrices using

$$d_{ij}^2 = s_{ii} + s_{jj} - 2s_{ij}. \tag{2.3}$$

The resulting dissimilarity matrix fulfills requirements for a distance: positiveness, a zero distance only to the point itself, symmetry, and triangle inequality.

PSD matrices and their distance counterparts correspond to local or non-linear metrics: topology of the space can vary from region to region. Therefore, by having only the pairwise training point distances available, distances between test points can be only approximately determined. Determining an unknown distance between two points would require computing a path integral between the points.

Mahalanobis distance Mahalanobis distances [116] are a more restricted class of distances than those obtained from general PSD matrices. Mahalanobis distances correspond to linear transformations of data, since the squared Mahalanobis distance, with $A = W^TW$,

$$d_{ij}^2 = (x_i - x_j)^T A(x_i - x_j) \tag{2.4}$$

can be represented as

$$\begin{aligned} d_{ij}^2 &= (x_i - x_j)^T W^T W(x_i - x_j) \\ &= (Wx_i - Wx_j)^T(Wx_i - Wx_j). \end{aligned} \tag{2.5}$$

Euclidean distance is a special case where the linear mapping is an identity matrix. A diagonal matrix specifies an adaptive Euclidean distance [166], where only scaling of the axes is possible. A general Mahalanobis distance has all properties that can be specified by a linear

mapping: axes of the data space can be rotated, scaled and used as linear combinations.

2.1.6 Neighborhoods

There are basically two ways for defining a neighborhood for a point. First, we can fix the number of neighboring points k we want to have in the neighborhood. This creates neighborhoods of differing spatial sizes, depending on local data density. Usually we want the neighbor relation to be symmetric. This can be arranged either by making points neighbors only if they both agree on this (in which case some points will have fewer neighbors than the intended k), or creating a connection if any point lists the other as a neighbor (number of neighbors grows for some points). The edges between neighbors can be weighted or unweighted. Weights are needed e.g. for computing geodesic distances. A problem of k-parameterization is the computational expense of finding the nearest neighbors.

Another way is to fix a spatial radius ϵ, and connect a point to all neighbors at a closer distance. In ϵ-neighborhoods points can have different numbers of neighbors. A neighbor relationship defined this way is symmetric. As all neighbors are about equally far away, edge weights do not bring much additional information [194], and are usually discarded.

Outliers become easily disconnected in ϵ-neighborhoods. The ϵ-neighborhood may also be a bad choice if the data has multiple natural scales. A value of ϵ that gives good results in a dense area may not represent sparser areas well, whereas using k nearest neighbors adapts to local conditions better. Data dimensionality, which affects the meaningfulness of neighborhoods anyway, must be taken into account especially when choosing ϵ, since the higher the dimension the less neighbors there will be inside a given radius.

Neighbor relationships can be hard or soft. In a hard neighborhood, two points either are neighbors or they are not. In a soft neighborhood, all points are neighbors, but the strength of the relationship varies, usually depending on the spatial distance via a kernel, most often a Gaussian. The kernel emphasizes closest neighbors by assigning them larger weights.

Defining hard neighborhoods is straightforward, whether using k or ϵ. Soft neighborhoods are more often parameterized by ϵ, which now becomes a kernel width parameter. Adaptive kernel widths, similar in effect to k-neighborhoods, can be created with soft k-parameterization. For example, the distance to the kth neighbor $x_i^{(k)}$ is used in [215] to determine local scale $\sigma_i^2 = \|x_i - x_i^{(k)}\|^2$, which is then used to weight connections as $w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma_i \sigma_j}\right)$. SNE and t-SNE (Sec. 2.4.3) determine kernel width so that the entropy of a Gaussian kernel is $\log k$. This amounts to using k as a kind of effective number of neighbors.

Neighbor relations are usually presented as a matrix, that has a nonzero entry for each existing connection between neighbors. The matrix can be equivalently interpreted as a graph, with (possibly weighted) edges between neighbors.

2.1.7 Embedding quality

There is no general agreement about what, if any, is a good representation of high-dimensional data with fewer dimensions. Different methods em-

ploy different cost functions, making a choice on what is worth preserving in the low-dimensional embedding. Since each method optimizes its own cost function, fair comparisons between methods are hard to make based on such cost functions. Therefore, several external quality criteria have been developed.

Development of the quality criteria has followed the development of DR methods. The early DR methods, like multidimensional scaling (Sec. 2.4.3), aimed at global preservation of distances, and also the quality of results was measured by global criteria. In 1990s, when the topology-preserving self-organizing maps [95] were at the peak of their popularity, the quality criteria concentrated on quantifying topology preservation. The current trend is to emphasize local neighborhoods, in DR methods as well as in quality criteria.

Global criteria Shepard diagrams [164] are a basic tool for assessing distance preservation. They are simple plots where pairwise distances realized in the embedding space are plotted as a function of the original pairwise dissimilarities. This is basically correlation of distances, and does not tell the whole truth about the embedding. An example from [19] is a mapping of an A-B-C triangle with edge lengths 2,3,4 to a straight A-B-C line with AB-distance 1, BC-distance 2 and AC-distance 3. Distance correlation is one, indicating a perfect mapping, yet we have completely lost the information about the points forming a triangle and not originally lying on a line. Several correlation-based quality measures have been suggested [68, 16, 19, 167].

Preserving topology Criteria for measuring topology preservation are more complex than the global criteria. Topographic product [7] is adapted from nonlinear dynamics. It measures preservation of neighbor orderings at different scales and combines the results into one formula. As scale terms are computed for all scales (numbers of neighbors) and all points, evaluating the criterion is computationally demanding. Topographic function [191] is computed using Voronoi regions of points to define neighborhoods, and is therefore best suited to clustering-based methods like SOM. Topographic function vanishes if topology is completely preserved, and positive and negative values measure the direction and severity of the mismatch between the embedding dimensionality and the intrinsic dimensionality.

Local criteria Rank-based criteria do not pay attention to pairwise distances, but only check whether sets of nearest neighbors are realized as they should in the embedding space. Different criteria can be classified in terms of their handling of hard and mild extrusions and intrusions [110]. In a hard k-extrusion, a point is missing from the k-neighborhood where it should be, and in a hard k-intrusion, it appears in a wrong k-neighborhood. Mild extrusions and intrusions occur when the point is included in the correct k-neighborhood, but its rank among the neighbors is wrong.

Local continuity metacriterion [28] combines mild intrusions and extrusions into one criterion. More common approach is to use two separate criteria. Trustworthiness and continuity (introduced in more detail below)

measure number and severity of hard intrusions and hard extrusions, respectively. Mean relative rank errors [109] behave similarly but include also mild intrusion and extrusions. [110] proposes two criteria, one for measuring intrusiveness or extrusiveness of the method, and another for assessing overall quality.

Many pairwise criteria are closely related to precision and recall [110]. Precision measures the proportion of correct answers among all answers to a query, and recall tells, which proportion of the possible correct answers was found. In DR embeddings, a correct answer could be a point which is placed in a correct k-neighborhood. Unlike the DR quality criteria, precision and recall do not consider the severity of rank mistakes.

Having two separate criteria makes it possible to describe the characteristics of different methods, which often implicitly emphasize avoidance of either extrusions or intrusions. They also open a way for creating controlled compromizes between the two properties. In spite of their symmetrical definition, the members of a criteria pair are not necessarily equally important. In [110] it is found that extrusive methods, i.e. those which may cause discontinuities by allowing more extrusions than intrusions, usually unfold manifolds better than intrusive methods. In [91] it is argued that trustworthiness, i.e. minimizing intrusions, is more important for data exploration than continuity.

Trustworthiness and continuity Trustworthiness and continuity [184, 91] are a popular pair of criteria for assessing neighborhood preservation. Newer work by the same authors [188], uses mean smoothed precision and recall instead, but we present the older and more heuristical trustworthiness and continuity here, since they were used in the studies on which this work is built.

Trustworthiness and continuity study numbers of correct and incorrect points in a neighborhood of a given size, taking into account how far from the neighborhood (in rank distance) a wrong point came from, or how far it should have been placed.

When studying trustworthiness at $\mathbf{x}_i$, other points are ordered by the distance from $\mathbf{x}_i$. Point ranks according to data space distance are denoted $r(\mathbf{x}_i, \mathbf{x}_j)$, and $\hat{r}(\mathbf{x}_i, \mathbf{x}_j)$ according to distances in the low-dimensional space.

Let set $U_k(\mathbf{x}_i)$ contain all points that are in the embedded neighborhood but should not be, and set $V_k(\mathbf{x}_i)$ the points that should be in an embedded neighborhood but are not. Then, for a k-neighborhood, trustworthiness and continuity, respectively, are defined as

$$T_k = 1 - A_k \sum_{i=1}^{N} \sum_{\mathbf{x}_j \in U_k(\mathbf{x}_i)} [r(\mathbf{x}_i, \mathbf{x}_j) - k] \tag{2.6}$$

and

$$C_k = 1 - A_k \sum_{i=1}^{N} \sum_{\mathbf{x}_j \in V_k(\mathbf{x}_i)} [\hat{r}(\mathbf{x}_i, \mathbf{x}_j) - k]. \tag{2.7}$$

The normalizing factor $A_k = \frac{2}{G_k}$, where

$$G_k = Nk(2N - 3k - 1), k < N/2, \tag{2.8}$$

$$G_k = N(N - k)(N - k - 1), k \geq N/2, \tag{2.9}$$

Table 2.1. A taxonomy of some dimension reduction methods.

	cost function			
(dis)similarity	**kernel eigendec.**	**Laplacian eigendec.**	**nonlinear dissimilarity**	**nonlinear similarity**
Euclidean distance	classical MDS [174, 175], PCA		MDS [98], Sammon [152], MDS-tradeoff [129], CCA [40, 76], CCA-tradeoff [185, 186]	
kernel-weighted neighborhoods	KPCA [158]	LE [8], linear LE [75]		SNE [78], t-SNE [182], SNE-tradeoff [187, 188]
paths through the data	Isomap [170]	diffusion map [32, 101, 126]	CDA [107]	
neighborhood + cost	MVU [203, 201], MVE [162]	LLE [148], linear LLE [74], HLLE [44]		

facilitates interpretation of the criteria by ensuring that the values are in the [0,1] range. Value one of trustworthiness and continuity is for perfectly successful mappings, and a perfectly random mapping gives values around 0.5.

2.2 A taxonomy of dimension reduction methods

Strikingly many DR methods are built from a part which applies a metric transformation or learns a data-based metric, and from a part which minimizes a cost function in the transformed space. Although not all methods neatly fall to this mould, division into a metric part and a cost function part serves as a useful framework for presenting the multitude of methods. This kind of division also brings forth the potential for new combinations, e.g. by using a non-linear cost function with a metric that is usually associated with an eigenfunction cost. Also [109] suggests a taxonomy that that separates metric and optimization, although they classify methods by the optimization algorithm, not by the cost function. Table 2.1 gives a taxonomy of some DR methods arranged in the metric-cost framework.

2.3 Metrics

2.3.1 Euclidean distance and its transformations

An often used dissimilarity is the Euclidean distance. In some DR methods it is used as such, but it is more common to form a matrix of Euclidean distances and transform it, e.g. with a suitable kernel function, to obtain new (dis)similarities. Often the role of the transformation is to emphasize local structure.

Gaussian or other kernels with a spherical or elliptical neighborhood are commonly used. Such kernels have a clear notion of locality, an area around a data point which is emphasized more than distant areas. The role of locality in kernels like $tanh$-kernel (Sec. 4.1.2) or polynomial is not as obvious as that in a Gaussian, and they are not much used in DR.

A kernel specifically designed for DR is proposed in [111]. The similarity of two points is based on the probability of observing a larger distance than the truly observed one, according to a multidimensional normal distribution and a threshold parameter (which controls the size of local neighborhood). With appropriate choices of the threshold and the dimensionality of the normal distribution this kernel can account for the norm concentration phenomenon [111].

Neighborhood probabilities
The probability p_{ij} of points $\mathbf{x}_i$ and $\mathbf{x}_j$ being neighbors can be computed by placing kernels at the points and normalizing. The kernel emphasizes local structure of data, and width parameter σ_i determines what is considered local.

Probabilities can be defined both in the high-dimensional space, computed from the raw data, or in the low-dimensional embedding space, in which case the point locations are optimized to satisfy a cost based on the neighborhood probabilities. By far the most common kernel is Gaussian, which gives neighborhood probabilities [78, 182] as

$$p_{j|i} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2\right)}{\sum_{h\neq i}\exp\left(-\|\mathbf{x}_i - \mathbf{x}_h\|^2/2\sigma_i^2\right)}. \tag{2.10}$$

If varying kernel widths are used, resulting probabilities are not symmetrical, and can be symmetrized by

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2}. \tag{2.11}$$

A modification with sometimes substantially improved performance is to use Student t-distributions [180], with neighborhood probabilities

$$q_{ij} = \frac{(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{h\neq l}(1 + \|\mathbf{z}_h - \mathbf{z}_l\|^2/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{2.12}$$

where α is a degrees-of-freedom parameter. The heavy tails of the t-distribution allow the points to move slightly farther from each other in the embedding than in the data space. This is necessary, because in the high-dimensional space, the distances concentrate, so that a point has many approximately equidistant neighbors. Realizing theăequidistance using a lower number of coordinates would force large numbers of points into same area, a phenomenon named crowding problem in [182].

Graph Laplacians
Many methods use neighborhood-based similarities to define weights of a graph, and use them to build so called graph Laplacian [31]. The Laplacian is a special matrix, that has theoretical connections to Laplacian operators defined on manifolds [9, 31]. There are several versions of the graph Laplacian [194], but the form usually used in dimension reduction is the unnormalized form, $L = D - W$. It is computed from edge weights W and a diagonal matrix D with entries $d_{ii} = \sum_j w_{ji}$. Defined this way, the matrix is real, symmetric and diagonally dominant, and thus guaranteed to be positive-semidefinite.

2.3.2 Paths through the data cloud

Geodesic distance
As opposed to the Euclidean straight-line distance, the geodesic distance follows the shape of a manifold. Approximate geodesic distances are obtained by finding shortest paths through the neighborhood graph. Dijkstra's [42] and Floyd's [53] algorithms compute shortest paths, Dijkstra's from one point to all points and Floyd's between all point pairs. Shortest-path approximations to geodesics may suffer from distortions due to holes, caused by non-uniform or sparse sampling, and from shortcuts, if the data has noise [183]. The graph should also be connected for geodesic distances to work well.

Diffusion distance
An approach less sensitive to noise is to regard the graph Laplacian as a Markov chain, and perform a random walk on it. The distance between two nodes is influenced by lengths of all paths between the nodes. Some paths might be due to noise, but they are averaged out by the hopefully more numerous paths that follow the manifold. One random-walk based distance which has been used in dimension reduction is diffusion distance [101, 126]. The transition matrix for the random walk is formed from a normalized graph Laplacian, and the distance can be computed from the eigenvectors and eigenvalues of this matrix.

2.3.3 Optimization using neighborhoods

Locally Linear Embedding (LLE) [148] works on the assumption that if a point can be reconstructed as a linear combination of its k neighbors in the data space, the same weights and same neighbors can reconstruct it in the embedding space. This is true if each local neighborhood is embedded using a linear mapping. The similarity matrix, which is naturally sparse, is found by optimizing the weights to minimize the reconstruction error.

Maximum Variance Unfolding (MVU) [203, 201] and Minimum Volume Embedding (MVE) [162] use nearest neighbor relationships to determine optimal similarities from data. A similarity matrix is learned so that some global property of the low-dimensional representation is optimized, while constraints are used to exactly preserve distances to the nearest neighbors. The global property varies depending on the method. MVU, as the name says, maximizes the variance of the results, which has the effect of unfolding curved manifolds. MVE tries to find representations so that the highest eigenvalues of the kernel matrix contain most of the energy, minimizing information loss in the final projection step of the method.

2.4 Cost functions

After the data has been represented in a suitable metric, a transformation to a low dimensionality is sought. The transformation is specified by a cost function. A simple alternative employed in many methods is the mean square error. Its minimization corresponds to eigendecomposition, either of a kernel matrix or of graph Laplacian. There are also methods that specify a nonlinear transformation.

2.4.1 Eigendecomposition of kernels

One simple way to obtain embedding coordinates after the similarity matrix is formed is to use eigendecomposition. Projections to the highest eigenvectors (those related to the largest eigenvalues) give the embedding locations for the data points.

If the kernel matrix is the covariance matrix of the raw data points, eigendecomposition performs principal component analysis (PCA). PCA finds a linear transformation of the data such that the directions of maximum variation become the new coordinate axes, in decreasing order of importance. Using other kernels than covariance corresponds to doing PCA in the metric presented by the similarity matrix. This method is known as kernel PCA (KPCA) [158]. In this sense all eigendecomposition methods can be considered variants of kernel PCA.

A separately developed method, that later was shown to be equivalent to PCA, is classical multidimensional scaling (MDS) [174, 175]. (The qualifier "classical" tells the method apart from the family of newer MDS methods which map data nonlinearly). It was formulated as a solution to the problem of finding absolute distances (a scale, hence the name) between points given a set of pairwise comparisons or relative distances. The same problem in a different formulation is to find locations for points in space given their pairwise dissimilarities. If the dissimilarity matrix represents Euclidean distances, the points can be exactly recovered by eigendecomposition [213]. Thus, classical MDS performs PCA, but formulated in terms of outer product of data points instead of the PCA inner product formulation [69].

A number of methods, originally not formulated using kernels, can be recast into the kernel framework [14]. These methods include LLE, Laplacian eigenmaps (see Sec. 2.4.2), MDS and Isomap.

Several other methods combine learning a method-specific similarity with finding the embedding by eigendecomposition. Classical MDS with geodesic distances, known as Isomap, is presented in [170]. MVU and related methods [203, 201, 162] decompose kernel matrices that have been optimized to maximize variance of the embedding or a related cost function. MVU is a popular method and has inspired several variants. In [195], requirement of exact neighbor distance preservation is relaxed, which reduces sensitivity to noise. [63] applies the MVU idea to data that is given as pairwise similar/dissimilar/unknown relationship constraints.

2.4.2 Eigendecomposition of graph Laplacians

Considerably larger number of methods obtain their embeddings from graph Laplacians. A graph Laplacian is a positive semidefinite matrix, such as kernel matrices are. However, it does not represent similarity, but is rather a special kind of dissimilarity matrix. Its PSD property is due to its construction, which makes the matrix diagonally dominant.

The non-diagonal entries contain negative edge weights, and therefore a smaller non-diagonal entry indicates larger similarity. This explains why decomposing Laplacians differs from the kernel eigendecomposition methods in that not highest but lowest eigenvectors (those related to the smallest eigenvalues) are used. The lowest vector is a constant, related to eigenvalue zero, and is excluded.

Embeddings using graph Laplacians were brought to awareness of the machine learning community by Belkin and Niyogi's work of Laplacian Eigenmaps (LE) [8], which not only introduced graph Laplacians as an effective and versatile tool for dimension reduction, but also led into reinterpretation of LLE as Laplacian-based [73, 11]. Graph Laplacians have earlier been used for spectral clustering, and their role in forming 2D representations for graphs has long been known as part of graph theory [65]. They have been applied in computational chemistry for drawing molecule structures [143].

Examples of other Laplacian-based methods include LLE [148], Hessian LLE (which replaces Laplacian with Hessian operator) [44], Neighborhood Preserving Embedding [74] (a linear approximation of LLE), Locality Preserving Projections [75] (a linear version of Laplacian Eigenmaps), and Diffusion Maps [32, 101, 126] (which are based on random walks but can be computed using Laplacians). Embeddings produced by LE or LLE can be further refined by a linear mapping to better preserve angles between data vectors [161].

2.4.3 Cost functions specifying a nonlinear transformation

MDS and variants

Term multidimensional scaling (MDS) refers to a whole group of methods, all of which try to find point locations so that Euclidean distances between the low-dimensional points match the given dissimilarities as accurately as possible. Their development was started from classical MDS, that is equivalent to PCA.

Later versions of MDS [164, 98] relax the classical MDS requirement that dissimilarities in the data should equal distances in the embedding. Instead, the dissimilarities can be transformed first. Transformations are specified by the cost function.

MDS is used in psychometry for handling results of pairwise comparisons, and more generally as a dimension reduction method. In psychometry, the goal is often to check if the pairwise observations can be explained by an assumption of a psychological space [19]. If an embedding using a certain distance measure matches the observed dissimilarities, the psychological space is deemed to obey that distance.

Many MDS cost functions, so called stress functions, are derivations of the mean square error used in classical MDS. General form for MDS stress functions [19], from which different costs can be formed by a suitable choice of $f(\cdot)$ and w_{ij}, is

$$s(\mathbf{z}) = \sum_{i<j} w_{ij}[f(\delta_{ij}^2) - f(d_{ij}^2(\mathbf{z}))]^2. \tag{2.13}$$

Some well-known cost functions are Stress ($f(\mathbf{z}) = \sqrt{\mathbf{z}}, w_{ij} = 1$), Squared stress ($f(\mathbf{z}) = \mathbf{z}, w_{ij} = 1$), and Sammon mapping ($f(\mathbf{z}) = \sqrt{\mathbf{z}}, w_{ij} = \delta_{ij}$). Sammon mapping [152] does not explicitly specify any local environment, but emphasizes short distances more than basic MDS does, and is in this sense a precursor for modern local dimension reduction methods.

CCA and variants

Instead of using data-space neighborhoods for defining distances, neighbor information can be introduced at the level of cost function. This allows defining neighborhoods in the embedding space, as is done in Curvilinear Components Analysis (CCA) [39, 40, 76]. The basic CCA and its derivatives use Euclidean distances. Also geodesic distances have been tried [107], under the name Curvilinear Distance Analysis (CDA).

Defining neighborhoods in the embedding space gives CCA its ability to unfold manifolds. Requiring accurate distance preservation inside an embedding-space neighborhood ensures that all points that are placed near each other, are truly data-space neighbors. But, nothing prevents data-space neighbors from spreading into different neighborhoods in the embedding space. The points that in the data space were brought near each other by manifold folding are allowed to move to different areas in the low-dimensional space, and thus the manifold can unfold.

For small distances ($\mathrm{d}(\mathbf{x}_i, \mathbf{x}_j) \leq \mathrm{d}(\mathbf{z}_i, \mathbf{z}_j)$) CCA minimizes a cost implementing unfolding,

$$E_u = (\mathrm{d}(\mathbf{x}_i, \mathbf{x}_j) - \mathrm{d}(\mathbf{z}_i, \mathbf{z}_j))^2 \cdot \mathrm{F}(\mathrm{d}(\mathbf{z}_i, \mathbf{z}_j), \sigma), \tag{2.14}$$

and for large distances, it uses a cost that implements projection to the unfolded manifold,

$$E_p = (\mathrm{d}(\mathbf{x}_i, \mathbf{x}_j)^2 - \mathrm{d}(\mathbf{z}_i, \mathbf{z}_j)^2)^2 \cdot \mathrm{F}(\mathrm{d}(\mathbf{z}_i, \mathbf{z}_j), \sigma). \tag{2.15}$$

σ is neighborhood radius; only points closer than that are considered, which is ensured by step function $F(\cdot)$. The optimization proceeds from global towards local scale by gradually decreasing σ.

Although the CCA cost is formulated using distances, it can be cast in a form that uses similarities instead, and related to similarity-based methods [111]. Representation using similarities highlights importance of the local neighborhood $F(\cdot)$, and explains why the seemingly distance-based CCA can behave more like locality-emphasizing methods than like MDS.

First versions of CCA used a neural network for vector quantization and building a description of data topology. This speeds up training, since only the code vectors of the quantization need to be mapped. Inspired by this idea, some other combinations of quantization and projection methods have been tried (competitive learning with a neighborhood preserving iterative algorithm [108, 106], neural gas with CCA and CDA [49, 48]). CCA itself can be, and has been, implemented without the quantization step.

SNE and variants

Stochastic neighbor embedding (SNE) [78] and related methods use cost functions that try to preserve the probability of two points being neighbors. The low-dimensional embedding is found by minimizing Kullback-Leibler divergence

$$C = \mathrm{KL}(P \| Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.16}$$

between high-dimensional neighborhood probabilities p_{ij} and low-dimensional probabilities q_{ij}, computed as in Eq. (2.10) and Eq. (2.12), respectively.

SNE computes the probabilities using Gaussians in both the high-dimensional and the low-dimensional space. A relatively new variant which

has rapidly become popular is t-SNE, or SNE using Student t-distributions [182], which uses Gaussians in the high-dimensional space but t-distributions for the embedding points. Originally t-SNE was meant especially for 2D and 3D visualizations, but it was later slightly modified to better adapt to different dimensionalities [180]. SNE and especially t-SNE have inspired a number of variants or related works [111, 25, 188], and SNE-like cost functions have been used in conjunction with other methods [181, 22].

Both SNE cost and its derivatives use squared Euclidean distances, but they can easily be adapted to use other distance metrics. In Chapter 3 we will use model-based similarities with t-SNE.

2.4.4 Trading off local quality for global quality

A number of approaches strive for a tradeoff between trustworthiness and continuity, or global and local quality.

Local and global constraints mean different things in the data space and the embedding space. Keeping points of a data-space neighborhood together in the embedding space requires notion of locality in the data space. Making sure that two data-space neighborhoods are not accidentally mapped on top of each other requires access to distant data-space points, and is thus a more global problem. In the embedding space, however, the latter problem is local: any two points in a local embedding-space neighborhood should originate from the same area in the data space. Such thinking has allowed development of methods with controlled tradeoff between local and global quality (Sec. 2.4.4).

Tradeoff methods have been built on MDS, CCA and SNE cost functions [129, 186, 188]. None of the cost functions optimizes trustworthiness or continuity explicitely, although SNE cost has a close connection to these criteria. Instead, the tradeoff is built between errors in the data space neighborhoods versus those in the embedding space neighborhoods. MDS-tradeoff uses either the data space or the embedding space distances as normalizing factors in the cost function. CCA-tradeoff explicitely formulates the neighborhoods in either space, and SNE-tradeoff uses either the data space or the embedding space probabilities as the distribution to be approximated by the KL-divergence.

How this relates to trustworthiness and continuity criteria can be understood by looking at the scales at which the two criteria operate. Continuity is predominantly a local property: to achieve high continuity, it is enough to obey local constraints, dictated by (dis)similarities to near neighbors (in the data space). This is why the continuity term looks at a data space neighborhood, and tries to minimize errors in it.

The continuity term is not affected by large scale mistakes. Two distant neighborhoods can be mapped on top of each other, as long as both are internally correct. To prevent such overlaps, global scale constraints are needed. Such constraints can be formulated as favoring mappings, where large data space dissimilarities become large embedding space distances. Alternatively, the constraints can punish mappings that turn large dissimilarities into small distances. This latter formulation gives the trustworthiness term in a tradeoff cost: a local neighborhood in the embedding space may only have points from a local data space neighborhood, that is, no distant data-space points are allowed.

MDS-tradeoff
Quality of MDS visualizations for dimension reduction has been analyzed in [129]. The cost function of Sammon mapping is seen as tearing error, minimization of which gives a continuous result. This is achieved by normalizing the square error by the original dissimilarities, which emphasizes small distances. On the other side, small flattening error (or high trustworthiness) can be achieved by emphasizing large distances, or normalizing by the output distances. From these, a cost function with a tunable tradeoff parameter λ

$$E = \frac{1}{N(N-1)} \sum_{i}^{N} \sum_{j=1, j\neq i}^{N} \Big(\lambda \frac{(D_{ij} - \delta_{ij})^2}{D_{ij}} + (1-\lambda) \frac{(D_{ij} - \delta_{ij})^2}{\delta_{ij}} \Big) \qquad (2.17)$$

provides a controlled tradeoff between the two errors.

CCA-tradeoff (Local MDS)
CCA tends to produce embeddings with high trustworthiness. The relationship of continuity and trustworthiness can be controlled by linearly combining a cost based on the embedding-space neighborhood with a cost that uses data-space neighbors. This is done in Local MDS [185, 186], with cost function

$$E = \frac{1}{2} \sum_{i \neq j} \left(\mathrm{d}(\mathbf{x}_i, \mathbf{x}_j) - \mathrm{d}(\mathbf{z}_i, \mathbf{z}_j) \right)^2 \times [(1-\lambda)\mathrm{F}(\mathrm{d}(\mathbf{z}_i, \mathbf{z}_j), \sigma_i) + \lambda \mathrm{F}(\mathrm{d}(\mathbf{x}_i, \mathbf{x}_j), \sigma_i)]. \qquad (2.18)$$

When $\lambda = 0$, the method works like the unfolding-half of CCA, and with $\lambda = 1$, the cost is MDS stress function, but only evaluated for the nearest neighbors.

SNE-tradeoff (Neighbor Retrieval Visualizer)
The SNE cost is studied in [187] from information retrieval perspective. If the neighborhood would be a step function, minimizing KL divergence from the data space neighborhoods to the embedding space neighborhoods would maximize average recall of neighborhood relationships. The SNE cost that uses Gaussian neighborhoods can be interpreted as a soft version of recall, as it takes into account not only the number of errors but also their size. Since continuity and recall are very similar, this observation explains why SNE tends to produce embeddings with high continuity.

A controlled tradeoff between continuity and trustworthiness is built in a method called Neighbor Retrieval Visualizer [187, 188] by introducing another cost term, that minimizes KL-divergence to the opposite direction, from the embedding space to the data space. This is a soft version of maximizing precision, a criterion related to trustworthiness. The total cost is

$$E = \lambda \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}} + (1-\lambda) \sum_{j \neq i} q_{ij} \log \frac{q_{ij}}{p_{ij}}, \qquad (2.19)$$

with p_{ij} and q_{ij} defined as in Eq. (2.10) and Eq. (2.12). This becomes SNE cost if $\lambda = 1$, and with λ set to zero tries to avoid showing wrong points together in the embedding space.

2.4.5 Alternative approaches

Aligned mixtures of local views
There is a large family of methods that are based on the idea of a mixture model [20, 169, 149, 217, 38, 189]. Several linear models (PCA, factor

analysis, tangent space, or other, depending on the method) are specified, each placed in a different location. The locations may be data point locations, or cluster centers, if the data has been quantized first. Low-dimensional coordinates for data are found by weighting coordinates, given by the local models, by the responsibilities of the respective models. Parameters of the local models and the global responsibilities are optimized together so that different local views are smoothly aligned.

Probabilistic methods
The focus of this section was on methods whose goal is formulated as a cost function, and all studies presented later in this work relate to cost-function based DR. There is also a growing group of DR methods, that are based on probabilistic latent variable models. A small number of latent variables, corresponding to the low-dimensional coordinates, is thought to generate the data via a model. The model can be linear, like in the probabilistic interpretation of PCA [173], or non-linear, like in the Gaussian process latent variable model (GPLVM) [103].

An appealing property of probabilistic models is that they provide a principled means for choosing model parameters. On the other hand, they introduce the extra problem of having only a mapping from the latent space to the data space available. Some external mechanism, like back-constraints for GPLVM [104], is needed to ensure that also the mappings from the data space to the latent space behave as desired. Further, inference in a probabilistic model is often computationally more demanding than optimization needed in non-probabilistic methods.

2.5 Supervised dimension reduction

2.5.1 Dimension reduction for classification

In the following, generic data points are indexed with h, i, j and l. Coordinates of point i in the data space are denoted as $\mathbf{x}_i$, and those in the low-dimensional space as $\mathbf{z}_i$. N is the number of data points. Formulas below are from the cited references, but with unified notation.

Maximizing classification accuracy
Classification accuracy is maximized in supervised dimension reduction both directly and indirectly. An indirect approach is taken by linear discriminant analysis [52], which sees each class as a cluster, and tries to keep classes well separated by using a criterion similar to those found in clustering [207]. More direct approaches build their cost functions on classification performance: Neighborhood components analysis [66] maximizes the accuracy of leave-one-out classification, and large margin KNN-classification is used in [202]. The latter two approaches have been used as cost functions for deep neural networks, and will serve as comparison methods for the supervised bottleneck classifiers in Sec. 5.3.

Linear discriminant analysis Linear discriminant analysis (LDA) [52], also known as Fisher's linear discriminant after its inventor, optimizes linear separability of classes in the low-dimensional space by maximizing between-class scatter B. At the same time, it tries to keep within-class scatter W at a certain level.This is achieved by building the cost function

to use relationship of within- and between-class variances and finding a linear transformation T to maximize $\mathrm{Tr}[(T^TWT)^{-1}T^TBT]$. T can be solved from a generalized eigenvalue problem. A different cost function realizing the idea of minimal within-class and maximal between-class scatter is used in [166].

If classes obey the assumption of Gaussianity, LDA works well. There are obviously many situations where Gaussianity does not hold, and variants of LDA have been developed with this in mind. For example, local LDA [168] computes LDA for each pair of points, allowing large errors for distant points. This allows a multimodal class to split into several clusters, although inside each cluster the LDA criterion keeps the points together. Also a kernelized version of LDA exists [6].

Neighborhood components analysis Neighborhood Components Analysis (NCA) [66] learns a discriminative Mahalanobis metric by maximizing accuracy of nearest neighbor classification. The cost function measures the expected number of correctly classified points, using soft neighbor assignments.

NCA tries to set neighborhood probabilities

$$q_{ij} = \frac{\exp(-\|\mathbf{z}_i - \mathbf{z}_j\|^2)}{\sum_{i \neq h} \exp(-\|\mathbf{z}_i - \mathbf{z}_h\|^2)}, \quad q_{ii} = 0 \tag{2.20}$$

in the low-dimensional space so that the number of correctly classified points

$$n_{\mathrm{ok}} = \sum_{i=1}^{N} \sum_{j:c^i=c^j} q_{ij} \tag{2.21}$$

(where c^i denotes the class of point i) is maximized. Classification accuracy is based on nearest neighbor classification. If a point is placed close to same-class points, probabilities of same-class points being neighbors go up. Equation (2.21) captures this idea by class-wise summing of neighborhood probabilities. Computational complexity is $O(N^2)$.

The original NCA seeks a linear transform, i.e. it finds $\mathbf{A}$ in $\mathbf{z} = \mathbf{A}\mathbf{x}$ to optimize Eq. (2.21). Its neural net implementation [151] looks for a more general mapping $\mathbf{z} = f(\mathbf{x}; w)$, optimizing network weights w.

Another method essentially equivalent to (linear) NCA is informative discriminant analysis [140, 139] . It can be seen as a generalization of LDA, and can also be interpreted as performing PCA in Fisher metric based on a discriminative model [140] (see Sec. 3.1.3).

Another cost function closely resembling NCA is suggested in [209]. Instead of global neighborhood probabilities, the cost uses neighborhoods defined in terms of a local metric. This setting creates a "chicken-and-egg dilemma" not met with global metrics: locality must be defined in order to minimize the cost and thus find the metric, but to define what is local, the metric must be known. In [209], this problem is solved by probabilistic formulation and an algorithm resembling expectation maximization.

Large Margin KNN Classification A cost function for having a large KNN-classification margin (LMNN) in the low-dimensional representation is presented in [202], and applied as deep network cost function in [123]. Like NCA, the original LMNN looks for a linear mapping, and the neural

network implements a nonlinear version. Minimizing the LMNN cost

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{l=1}^{N}\eta_{il}\gamma_{ij}\cdot\max(0,1+\|\mathbf{z}_i-\mathbf{z}_l\|^2-\|\mathbf{z}_i-\mathbf{z}_j\|^2) \tag{2.22}$$

requires that the distance from point i to an other-class point j must be greater than the largest distance to k nearest same-class points (indexed with j), with margin of one.

Comparisons to k same-class neighbors and all other-class neighbors would make the computational complexity $O(kN^2)$. Therefore, only m other-class neighbors from each class are used, lowering the complexity to $O((C-1)kmN)$ (with C classes). Points to be used as neighbors are indicated by binary variables η_{il} and γ_{ij}. η_{il} equals one if point l is a chosen same-class neighbor for point i, and γ_{ij} encodes relationships to the other-class points in the same way. As other-class neighbors from all classes are needed, LMNN cost is sensitive to the number of classes.

Matching class-conditional distributions
In Parametric Embedding [62], a good supervised metric is assumed to be such which collapses all points in a class into one point. This ideal situation is presented by neighborhood probabilities (1 for same-class points, 0 for other-class points). In the low-dimensional space, the neighborhood probabilities are defined as in Eq. (2.10), but the locations of points are restricted to be $\mathbf{z}=\mathbf{A}\mathbf{x}$. $\mathbf{A}$ is found by minimizing Kullback-Leibler divergence between the ideal distribution and the neighborhood probabilities. This method has been interpreted as a supervised version of SNE.

A resembling idea for binary classification is presented in [196], where two Gaussian distributions are compared with KL-divergence. The goal, keeping all points of one class together, is encoded as so called ideal kernel $K_{\text{ideal}}=Y^TY$ (Y is a matrix of binary classifications), that becomes the covariance of one of the Gaussians. Covariance of the other Gaussian is formed from the input points and the distance metric to be learned, $K=X^TAX$. Then A is found so that the KL-divergence between the two Gaussians, with covariances K and K_{ideal}, is minimized.

The match of distributions, measured by KL divergence, is also used in [85]. Conditional distribution $p(c_k|x_n)$ in the data space is matched against a mixture of (unit variance) Gaussians, one for each class, in the embedding space. This tends to place points of same class close together, but also places two classes close to each other if they share many points. Also this method can be considered a supervised generalization of SNE; it reduces to SNE in the special case where each object forms its own class.

Other approaches
Supervision can be implemented as a cost function, which combines costs for preserving the structure of the input data properly, and for preserving the structure related to class predictions. A tuning parameter controls the respective importance of the two terms. Some supervised variants of autoencoders (Sec. 5.1.1) use this strategy. It has also been used to supervise GPLVM [178], using LDA cost as a prior term.

Sometimes supervision is introduced by ignoring the roles of inputs and targets, and simply using all variables as inputs. This approach is adopted in class-augmented PCA [137], and supervised probabilistic PCA [214] behaves equivalently [216], although the model has separate input and target variables.

Although this approach does allow information from the target vari-

ables to be included, ignoring the special role of the targets may lead into problems. The goal of supervised dimension reduction is to find low-dimensional representations that have a meaningful relationship with the target variables. If the nature of this relationship is not considered at all in the method, consistently getting good results would be surprising. Also, this approach suffers heavily from increasing input dimensionality. One target variable among three inputs may make a difference, but one among a thousand hardly will.

Class memberships can be emphasized by modifying the pairwise distance matrix, as do [60] and [193]. Both work by first computing a pairwise distance matrix in unsupervised fashion, and then applying a nonlinear transform to it. This transformation reduces distances for same-class points and increases those between classes. The transforms used are ad-hoc, but the general idea of using a metric which takes class memberships into account is not that different from many other better-motivated methods.

2.5.2 Dimension reduction for regression

Dimension reduction of regression (DRR) deals with a specific subproblem of supervised dimension reduction: to find low-dimensional representations such that a regression target can be reliably predicted based on the low-dimensional representation only.

Many DRR methods have their roots in multivariate statistics and have developed as a separate branch of research, distinct from the classification-oriented supervised dimension reduction methods, which are more familiar in the machine learning community. Ideas from unsupervised dimension reduction are sometimes applied in DRR problems, but the opposite is seldom true.

One central concept for DRR is that of sufficient subspace. Given such a subspace, predictions about the target variable can be made as accurately as using the whole space. Because of this, DRR is sometimes called sufficient dimension reduction (not to be confused with work of same name [64], but about a related topic of doing DR on co-occurrence data).

We will not attempt to cover the whole field of DRR, but introduce those methods which will be used in Chapter 5. The following descriptions are brief, and mostly follow the flow of implementation rather than that of theoretical derivations. See respective references for more thorough treatment.

Notation We refer to the high-dimensional input variable as $\mathbf{x}$, to the regression target as $\mathbf{y}$, and to the d-dimensional representation sought by the DRR methods as $\mathbf{z}$. Spaces where these vectors live are called x-space, y-space and z-space, respectively. Subscripts of boldfaced variables denote numbered data points ($\mathbf{x}_i$), and when necessary, individual elements of vectors are shown as subscripted non-boldfaced letters (x_i). If not mentioned otherwise, d=2. $\mathrm{Tr}[\cdot]$ is the matrix trace operator. Gaussian kernels are abbreviated as

$$\mathrm{Gauss}(\mathbf{a}_i, \mathbf{a}_j; \sigma) = \exp\left(-\frac{\|\mathbf{a}_i - \mathbf{a}_j\|^2}{\sigma^2}\right). \tag{2.23}$$

$(A)_{ij}$ is the element from row i, column j in matrix A.

Kernel dimension reduction
Kernel dimension reduction (KDR) [58] uses a cost function, which goes to zero if x and y are conditionally independent, given the low-dimensional variable z. This means that z has captured all information in x, and can thus be used instead of x for predicting the values of y.

The KDR cost function

$$\mathrm{Tr}[\mathbf{G_y}(\mathbf{G_x^B} + N\epsilon_N\mathbf{I_N})^{-1}] \tag{2.24}$$

uses two Gram matrices, $\mathbf{G_x^B}$ and $\mathbf{G_y}$, both built using Gaussian kernels. One is computed in y-space

$$(\mathbf{G_y})_{ij} = \mathrm{Gauss}(\mathbf{y}_i, \mathbf{y}_j; \sigma_y), \tag{2.25}$$

and does not change during the optimization. The other Gram matrix

$$(\mathbf{G_x^B})_{ij} = \mathrm{Gauss}(\mathbf{B}^T\mathbf{x}_i, \mathbf{B}^T\mathbf{x}_j; \sigma_x), \tag{2.26}$$

is likewise built with a Gaussian kernel but with the input points projected to the dimension reduction subspace using $\mathbf{B}$. Both matrices are centered by multiplying by the centering matrix $\mathbf{H} = \mathbf{I_N} - \frac{1}{N}\mathbf{1}\mathbf{1}^T$ (where $\mathbf{1}$ is a vector of ones) from both sides. Mapping $\mathbf{B}$ is found so that Eq. (2.24) is minimized. Minimization of $\mathbf{B}$ is carried out ensuring that $\mathbf{B}$ is orthogonal. We use simulated annealing for optimization. After finding $\mathbf{B}$, the low-dimensional points are computed from

$$\mathbf{z}_i = \mathbf{B}^T\mathbf{x}_i. \tag{2.27}$$

In addition to $\mathbf{G_x^B}$ and $\mathbf{G_y}$, KDR cost function depends on a regularization parameter ϵ_N, and two deviation parameters, σ_x and σ_y. Since ϵ_N and the deviations have similar effect, ϵ_N can be fixed (we use ϵ_N=0.1) and only the deviations chosen [58].

Manifold kernel dimension reduction
Manifold KDR (mKDR) [127] is an extension of KDR to situations where the input data lies on a nonlinear manifold. Manifold KDR opens the manifold structure by Laplacian eigenmaps (LE) [8] as a preprocessing step. When LE is used as a dimension reduction method, only a few lowest eigenvectors are used. Such preliminary dimension reduction can be done also when LE is used in mKDR.

After finding the LE features of data, mKDR minimizes the KDR cost function Eq. (2.24) in this feature space. Now the KDR cost is formulated including the LE eigenvectors $\mathbf{U}$,

$$\mathrm{Tr}[\mathbf{G_y}(\mathbf{U}^T\mathbf{\Omega}\mathbf{U} + N\epsilon_N\mathbf{I_N})^{-1}], \tag{2.28}$$

with $\mathbf{\Omega}$ the target of optimization. $\mathbf{\Omega}$ is restricted to be positive semidefinite.

Parameters needed for mKDR are neighborhood size c for LE, and ϵ_N, which is used to regularize the KDR x-space kernel. Finding $\mathbf{\Omega}$ which minimizes Eq. (2.28) is done by the projective gradient method [127]. It consists of successive steps of gradient descent and projection of $\mathbf{\Omega}$ to the cone of positive semidefinite matrices (which amounts to removal of eigenvectors that correspond to negative eigenvalues).

Sliced inverse regression

Use of inverse regression to find subspaces is based on the assumption that the mean $E(\mathbf{x}|\mathbf{y})$ lies at the same subspace as $E(\mathbf{y}|\mathbf{x})$. The former is easier to estimate, since $\mathbf{y}$ is usually much lower-dimensional than $\mathbf{x}$. Sliced inverse regression (SIR) [113] estimates $E(\mathbf{x}|\mathbf{y})$ by dividing the range of $\mathbf{y}$ into slices, and computing mean $\mathbf{m}_i$ for each slice. Covariance

$$\mathbf{V} = \sum_{i}^{S} \frac{p_i}{N} \mathbf{m}_i \mathbf{m}_i^T \tag{2.29}$$

is built by weighting each slice by the proportion of samples falling into that slice. Directions of maximum variance, given by the d highest eigenvectors $\mathbf{u}_k$ of $\mathbf{V}$, are used as the dimension reduction subspace.

As a preprocessing step, the data is whitened by multiplying with a whitening matrix $\mathbf{M}$ (inverse of square root of data covariance), and the original scale is returned by using

$$\beta_k = \mathbf{u}_k^T \mathbf{M} \tag{2.30}$$

as the low-dimensional points.

In [113] it is recommended to divide the range of $\mathbf{y}$ into slices so that all slices have approximately equal number of samples, and we use this strategy in our implementation. For two-dimensional targets, we divide the ranges for each target into slices separately. This results in some slices having fewer points than the others, especially near the ends of the target ranges.

Covariance operator inverse regression

Covariance operator inverse regression (COIR) [93, 94] is based on eigendecomposition of inverse regression covariance matrix. Although based on the idea of inverse regression like SIR, COIR does not need slicing, since it uses covariance operators for both the data and the targets. This makes it better suited for use with multivariate responses. COIR also avoids the strong assumptions of SIR about the distribution of $\mathbf{x}$. Main benefit of COIR over KDR is that it has an analytical solution.

COIR estimates the inverse regression covariance matrix using two Gram matrices, $\mathbf{K_x}$ and $\mathbf{K_y}$. They are formed by computing similarities from a training point i to all other training points j using Gaussian kernels, separately in x-space and y-space,

$$(\mathbf{K_x})_{ij} = \mathrm{Gauss}(\mathbf{x}_i, \mathbf{x}_j; \sigma_x), \tag{2.31}$$

$$(\mathbf{K_y})_{ij} = \mathrm{Gauss}(\mathbf{y}_i, \mathbf{y}_j; \sigma_y). \tag{2.32}$$

These matrices can be thought as covariances[1] of feature matrices $\mathbf{\Phi_x}$ and $\mathbf{\Phi_y}$,

$$\mathbf{K_x} = \mathbf{\Phi_x}^T \mathbf{\Phi_x}, \quad \mathbf{K_y} = \mathbf{\Phi_y}^T \mathbf{\Phi_y}. \tag{2.33}$$

$\mathbf{K_x}$ and $\mathbf{K_y}$ are used for estimating inverse regression covariance, whose d highest eigenvectors β are found from the generalized eigenvalue problem

$$\frac{1}{N} \mathbf{K_y} (\mathbf{K_y} + N\epsilon_N \mathbf{I_N})^{-1} \mathbf{K_x} \beta = \lambda \mathbf{K_x} \beta. \tag{2.34}$$

[1] Correction: This part was sloppily implemented and explained in the author's earlier work [131], where the feature vectors were built using Gaussian kernels, and their product was used as the Gram matrix. The inadvertent modification led to a version of COIR formed with similar idea as simplified kernel SIR [212]. The modification corresponds to squaring the kernel spectrum [29], and should not change the directions which COIR can find, although it changes the optimal parameter values.

From β and features $\mathbf{\Phi_x}$ we find the basis vectors $\mathbf{b}$ of the dimension reduction subspace, and the projections $\mathbf{c}$ of the training points to them

$$\mathbf{b} = \mathbf{\Phi_x}\beta, \quad \mathbf{c} = \mathbf{\Phi_x}\mathbf{b}. \tag{2.35}$$

Coordinates for the test points are obtained by projecting the feature space representation $\mathbf{\Phi_x^*}$ of the test points to the basis vectors $\mathbf{b}$

$$\mathbf{c}^* = \mathbf{\Phi_x^*}\mathbf{b}. \tag{2.36}$$

3. Metrics bridging models and dimension reduction

Since many DR methods are built by combining a suitable metric and a cost functions, their behavior is easily changed by changing the metric. In this chapter, we use different metrics to encode results of machine learning models ("a model" is understood in a wide sense to include also clustering methods). Combinations of a model and a DR method can be seen from two perspectives, depending whether the model or the DR result is seen as important. On the one hand, we can visualize modeling results (e.g. a clustering) or internal presentations of a model (e.g. feature space vectors) using DR. On the other hand, we can use a model as a preprocessing step to highlight those properties of data that we are most interested in when reducing the dimensionality.

In Sec. 3.2 we use a DR visualization as a tool for studying the results of hierarchical clustering. DR representations of hierarchical structures are space-efficient, and allow both showing the nestedness of structures and giving information about distances between the clusters. Such use of DR draws new attention to the distance-based DR methods, which advent of neighborhood-based methods has made unpopular.

In Sec. 3.3 we concentrate on the other viewpoint, by using a model to introduce supervision into unsupervised DR methods. In the literature the idea has already been explored for both generative and discriminative models (see 3.1.3). The appeal of this approach lies in easiness of implementation and in versatility of modeling tools as compared to existing supervised DR methods. We study this idea specifically in context of multilayer perceptron models. Internal representation such models is easily accessible through network weights, and this makes it possible to do DR in the feature space of the model. The more general approach, to which we compare our results, is based on changes in model predictions, and is computationally heavy. Our experiments show than in the case of neural network models the simple idea of using feature space metric can give equally good results.

This chapter presents updated versions of earlier work in [132, 135].

3.1 Metrics that describe results of modeling

3.1.1 Hierarchical clustering and cophenetic distance

Hierarchical clustering (HC) creates a hierarchical description, that can be cut at different levels to obtain partitionings with different granularities. A hierarchy describing the structure of data can be built in top-down or bottom-up fashion. The latter, agglomerative hierarchical clustering,

is more commonly used, and that is the form we describe here. To start with, each data point is placed in its own cluster. At each step, pairwise distances between all clusters are calculated, and the two clusters with the minimum distance are merged into one cluster. This process is continued until only one cluster remains. Results of the merging process are presented in the form of a binary tree, which can be cut to clusters based on criteria for the mutual distance or the number of clusters.

The gist of the HC algorithm is to define a way of measuring the distance between two clusters. This determines what kind of clusters will be formed. There are various methods, referred to as linkage methods, for determining the inter-cluster distance. Some common ones are single linkage (shortest pairwise distance), complete linkage (longest pairwise distance), average linkage (average distance) and Ward's method. Ward's linkage [197] minimizes the cost of each new grouping, measured by error sum of squares.

The hierarchical description of data which is given by HC can be used to define a metric, called cophenetic distance [72]. The metric represents the cluster structure in a form of pairwise distances of training points. During the merging process, the linkage method is used to measure the distance between any two clusters. When clusters A and B, whose distance is δ, are merged into one cluster, the cophenetic distances between all $a \in A$ and all $b \in B$ are set to δ. When HC results are represented by drawing dendrograms, the height of the link where the two points are first joined shows their cophenetic distance. As the cophenetic distance is only defined between two training data points, it cannot be used with methods that need to compute distances to other points (e.g. to codebook vectors in self-organizing maps).

3.1.2 Feature space of a one-layer feedforward network

A multilayer perceptron network (MLP) with one layer can be interpreted as a simple feature extractor. The hidden layer learns features relevant for the prediction task, and the output layer performs linear classification or regression in the feature space provided by the hidden layer. Since the features are learned for a supervised task, they contain information relevant for model predictions. Euclidean distances between such feature vectors provide a simple supervised metric.

The hidden layer consists of H $\tanh$-units. The output of the hth hidden unit is given by

$$\mathrm{hid_h}(\mathbf{x}) \quad = \quad \tanh\left(b_h + \sum_{i=1}^{D} w_{ih} x_i\right) \tag{3.1}$$

(notation: D-dimensional input vector $\mathbf{x}$ with elements x_i, weights w_{ih} from input x_i to hidden unit h, and biases b_h for hidden units).

A binary classifier has one logistic output that gives the probability of class 1. A multiclass classifier has one linear output unit per class, and their results are turned into class probabilities using softmax. Networks performing regression have one or many linear output units. Output of a binary classifier is

$$p(c = 1|\mathbf{x}) = \mathrm{logsig}(L(\mathbf{x})) = \mathrm{logsig}\Big(b' + \sum_{h=1}^{H} w'_h \mathrm{hid_h}(\mathbf{x})\Big) \tag{3.2}$$

(notation: weights w'_h from hidden unit h to output, and output bias b'; L

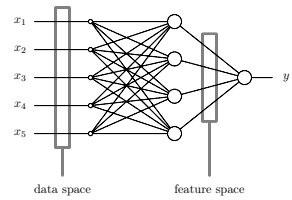


Figure 3.1. A one-hidden-layer MLP. The hidden layer extracts features, so that its outcomes can be interpreted as a feature space vector.

is used as an abbreviation for the output of the linear part of the output layer).

The feature vector for data point x is a vector of the outcomes from the hidden units (illustrated in Fig. 3.1)

$$f(\mathbf{x}) = [\mathrm{hid}_1(\mathbf{x}), \ldots, \mathrm{hid}_\mathrm{H}(\mathbf{x})]. \tag{3.3}$$

3.1.3 Discriminative Fisher metric

Fisher kernel and its derivatives are based on fitting a model to data, and describing the data in terms of change rates of model parameters. Metrics based on parameter changes were first built for generative models, and later generalized for the discriminative setting. For an MLP model, a discriminative Fisher metric contains much of the same information as a metric based on the feature space, but it also takes into account how the features are weighted in predictions.

Fisher information in generative models

A kernel for improving classification can be obtained by working in the gradient space of a generative model [86]. So called natural gradient describes changes, as function of model parameter θ, along a manifold where each point represents one possible model. Natural gradient $\phi = I^{-1}U$ can be computed from Fisher score $U = \nabla_\theta \log P(X|\theta)$, that describes the contribution of θ-model in generating the sample X, and from Fisher information matrix $I = E[UU^T]$. This leads to definition of Fisher kernel $K = \phi^T I \phi = U^T I^{-1} U$ as a similarity measure between the data points. For small distances, (Euclidean) distances in the natural gradient space can be approximated by Kullback-Leibler divergences between the models.

Fisher kernel is a maximum a posteriori approximation to a more general Bayesian approach called mutual information kernel [160]. Mutual information measures how much information two data points share through so called mediator distribution, that determines how strongly prior and posterior distributions should be weighted when forming the kernel. This corresponds to presenting data points by their likelihoods under all possible models, which are then integrated out to give total similarities between points.

Fisher information is used in [100] to define a kernel for model parameters. When a posterior for parameters has been found, the parameters can be integrated out to yield a kernel for data. The approach is general but difficult to compute.

A related approach, based on generative models but not using Fisher information, is to derive a metric from posterior probabilities of a mixture model. One such metric is presented in [172]. It is a weighted sum of several components, each of them a Mahalanobis metric, which are described

by covariances of the corresponding Gaussians. The metric is local and therefore requires path integrals if, a distance between faraway points is needed. A related metric [146] measures the changes in generating densities, and captures the structure of the data with approximate of geodesic distances.

Fisher information in discriminative models
The idea of the Fisher kernel can be generalized to discriminative models, and different versions have been suggested. We will apply the metric by [142], so called learning metric (LM), in experiments in Sec. 3.3.

The observation that the Fisher kernel can be interpreted as a feature extractor leads into definition of a more general kernel in [177]. A kernel that minimizes linear classification error in Fisher kernel feature space is sought. As the approach is based on log-odds of classes, it only works for binary classification.

The idea of [142] is to define an analogue of the Fisher metric for an auxiliary variable c. Change $d\mathbf{x}$ of variable $\mathbf{x}$ is considered the more meaningful the more it changes the auxiliary variable. The auxiliary variable can be the class probability in binary classification or a regression target, so the approach is more general than that of [177]. Changes in c are measured by Kullback-Leibler divergence. For small $d\mathbf{x}$, KL divergence is symmetric and can be used as a local metric. Thus, squared distances are computed as

$$d^2(\mathbf{x}, \mathbf{x}+d\mathbf{x}) \equiv D_{KL}(p(c|\mathbf{x}), p(c|\mathbf{x}+d\mathbf{x})) = d\mathbf{x}^T \mathbf{J}(\mathbf{x}) d\mathbf{x}, \tag{3.4}$$

where $\mathbf{J}(\mathbf{x})$ is Fisher information for a generic model $p(c|\mathbf{x})$ (the model can have parameters θ and should more accurately read $p(c|\mathbf{x}, \theta)$; we omit θ for brevity)

$$\mathbf{J}(\mathbf{x}) = \mathrm{E}_{p(c|\mathbf{x})}[(\nabla_{\mathbf{x}} \log p(c|\mathbf{x}))(\nabla_{\mathbf{x}} \log p(c|\mathbf{x}))^T]. \tag{3.5}$$

3.2 Visualizing hierarchical clustering

This section shows an example of using cophenetic distances in Sammon mapping visualizations, and discusses the combination as a visualization tool. The USPS example was first used in a work that studied assessing reliability of DR visualizations of hierarchical structures [132].

3.2.1 Cophenetic Sammon mapping and clusters in USPS data

We use cophenetic distances provided by the HC hierarchy tree as dissimilarities in Sammon mapping. Cophenetic distances have earlier been used for cluster-emphasizing MDS visualization [3].We also tried cophenetic distance matrix with t-SNE (not shown), but that combination tended to create so strongly concentrated clusters that the visualizations were hard to read.

Fig. 3.2 shows an example where 1000 random samples from the USPS data set have been clustered using HC with Ward's criterion. The visualization clearly shows that although numbers are often clustered according to their semantics, the ten digit classes do not form the most natural clusters, at least not for this clustering criterion.

The locations of points are determined by the DR algorithm based on the high-dimensional clustering, and a clustering based on the 2D coor-

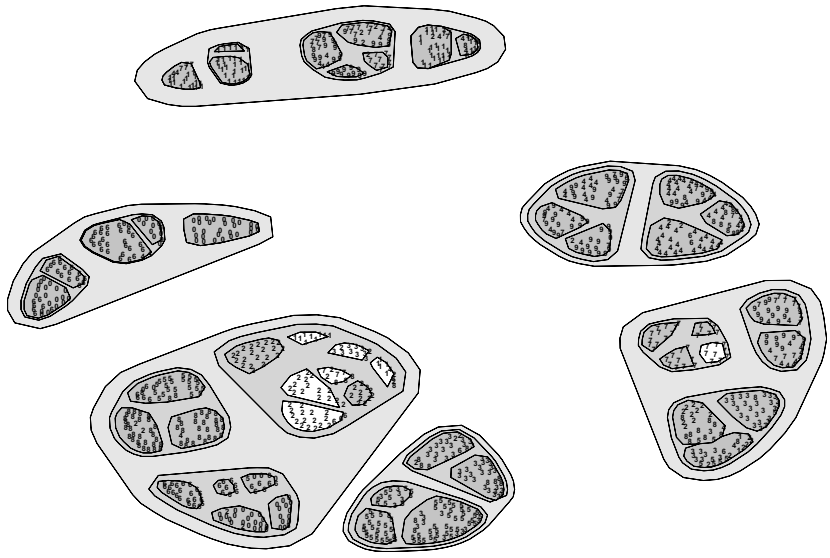


Figure 3.2. HC results of USPS data (1000 points) visualized using Sammon mapping with the cophenetic distance. The white areas inside the gray regions show clusters where the cluster borders, which are based on the 2D result, do not correctly show the high-dimensional clustering result.

dinates is used to draw boundaries. In ideal case, the user should understand the high-dimensional clustering result by looking at the 2D visualization – that is, any conclusions she draws about the clusters should be the same as conclusions based on the dendrogram showing the HC result. The visualization highlights areas where the 2D cluster boundaries differ from the dendrogram (for details of the criterion used, see [132]). Generally, the 2D result reliably shows the highest levels of the hierarchy that was derived from the high-dimensional data. When looking at smaller and smaller clusters, differences emerge. When the data is divided into 50 clusters (lowest level shown with the cluster borders), unrelated points are occasionally shown in the same cluster.

3.2.2 Properties of cophenetic Sammon visualizations

Using cophenetic distances with Sammon mapping tends to create structures with rounded shapes, resembling results of some network visualization algorithms which are designed for clusters [128]. Compared with such methods, cophenetic distance organizes the cluster shapes in a hierarchical fashion.

Hierarchically organized circles have been applied in visualizations also outside DR field [18, 171]. There are several ways to visualize hierarchies, mostly by different tree shapes, nested areas or repeated divisions of space. Nested circles are one variant of nested area representation. In comparison between different hierarchical visualizations in [121], nested-circle methods are found to make good use of the image area available, while having space for labels and allowing labels to be oriented the same way for readability. Also cophenetic Sammon visualizations have these properties. They differ from the basic nested circle representation in that

the distances between clusters can vary. This creates some empty space in the results, but makes cluster distances informative about node heights of the HC hierarchy tree.

Although Sammon mapping and other distance-based DR methods currently fall in shadow of locality-oriented methods, they can prove useful with metrics that already encode locality information. When a clustering-based metric is used, the metric structure is what we want to show. Therefore a cost function which tries to preserve distances can be a right choice, while a locality-emphasizing method like t-SNE may end up over-emphasizing cluster information. This may explain the better readability of Sammon visualizations compared to t-SNE, when cophenetic distances are used.

3.3 Comparison of feature space and LM visualizations

The idea for using MLP feature spaces in DR visualizations was first presented in [135], but that study failed to quantify the results, and also lacked a comparison to the closely related learning metric. This section repeats the most relevant experiments of [135] with different data sets and a more representative set of DR methods. We also compare metrics obtained from the MLP feature space with those obtained using the learning metric of MLP predictions.

LM is theoretically well-motivated. Its main drawbacks are heavy computation, and suitability only to models with scalar predictions. MLP feature space is readily available for multiple response models. Distances are fast to compute, since the Euclidean metric can be used. On the other hand, there are no guarantees on the feature space structure. The model may have learned some irrelevant features, which would be down-weighted in the outcome, but are present in the feature space.

The learning metric has been used for improving SOM visualizations in [141]. Both the learning metric and the MLP features have also been applied in supervised clustering, where the goal is to find clusters which are homogeneous with respect to a target variable. Discriminative clustering [92] performs k-means in the learning metric. Use of the MLP feature space is not tied to any specific clustering algorithm. It is applied in finding and describing class-homogeneous subpopulations in [136].

3.3.1 Fisher information in a binary MLP classifier

The learning metric for an MLP model can be calculated (as in [136]) by using the MLP output Eq. (3.2) in place of $p(c|\mathbf{x})$ in Eq. (3.5). Now, the auxiliary variable c is the prediction of the MLP classifier. In case of a binary classifier, expectation Eq. (3.5) becomes simply a sum of two terms, one for each class. We also need to compute gradients of network outcomes w.r.t. network inputs. For the binary classifier, the elements of the gradient vectors for classes 1 and 0 are computed using chain rule,

$$\frac{\partial}{\partial x_i}\log(\mathrm{logsig}(L)) = \frac{e^{-L}}{1+e^{-L}}\cdot\frac{\partial L}{\partial x_i} \tag{3.6}$$

(3.7)

$$\frac{\partial}{\partial x_i}\log(1-\mathrm{logsig}(L)) = -\frac{1}{1+e^{-L}}\cdot\frac{\partial L}{\partial x_i} \tag{3.8}$$

with the derivative of the linear part

$$\frac{\partial L}{\partial x_i} = \sum_{h=1}^{H} w'_h \Big(1 - \tanh^2(b_h + \sum_{i=1}^{D} w_{ih} x_i)\Big) w_{ih}. \quad (3.9)$$

3.3.2 Experiments

We use Euclidean distances in the MLP feature space (MLPfeat) and the learning metric for forming (dis)similarity matrices for five DR methods. We try the methods on five data sets: WDBC, TicTacToe, USvotes, Pima and Arcene (see Appendix A for details). Due to the very high original dimensionality of Arcene data, only its 500 highest eigenvectors (those related to the largest eigenvalues) are used. Other data sets are used in their original dimensionality. All MLP models have 20 hidden units.

The learning metric is local, that is, computing a distance between two points requires computing a path integral over the metric. Following [142], we use a piecewise linear approximation for this integral. The path between points is divided into 3 (MLPfisher3) or 10 intervals (MLPfisher10), and Fisher information at first point of an interval determines the metric for the whole interval.

We compare the metrics in creating 2D visualizations. The DR methods used are kernel PCA, Sammon mapping, t-SNE, Laplacian eigenmap (LE) and CCA. Kernel PCA corresponds to eigendecomposition of the model-based similarity matrix; no external kernels are used. The list of DR methods was chosen to contain both global and local methods. Methods using geodesic distances or another way for explicitly following manifolds are not considered, because we see no reason to assume that the model feature space vectors or predictions would form a manifold.

KPCA, Sammon mapping and CCA do not have tuning parameters (except for the schedule for decreasing CCA neighborhood size, for which we use the default values of SOM Toolbox for Matlab). T-SNE and LE require the neighborhood size. Values $k = 2j, j = 1 \ldots 25$ are tried for both of them. With methods that do not give deterministic results (Sammon, CCA, t-SNE) the method is run 10 times with each parameter value, and the result with the best cost function value is used.

3.3.3 Results

Fig. 3.3 shows an example of visualizations with the two supervised metrics, for t-SNE and two of the data sets (USvotes and TicTacToe). Compared with raw data, both supervised metrics result in more pronounced separation into classes.

A supervised visualization should clearly show the class structure of the data. Therefore KNN-classification accuracy is an appropriate numerical criterion for evaluating the results. Neighbor numbers from 1 to 20 are tried, and for each method, the best classification accuracy is recorded. For methods with neighborhood size parameters, the best result is shown. The numerical results are summarized in Fig. 3.4.

In the results we see a pattern which is common when comparing machine learning methods: results depend on the data set, and in our case, also on the dimension reduction method used. No clear order of superiority can be established between MLPfeat and MLPfisher. On the other hand, this is not a complete non-result. We expected the theoretically well-motivated, computation intensive MLPfisher to perform consistently better than MLPfeat, but that does not seem to be the case. That the more

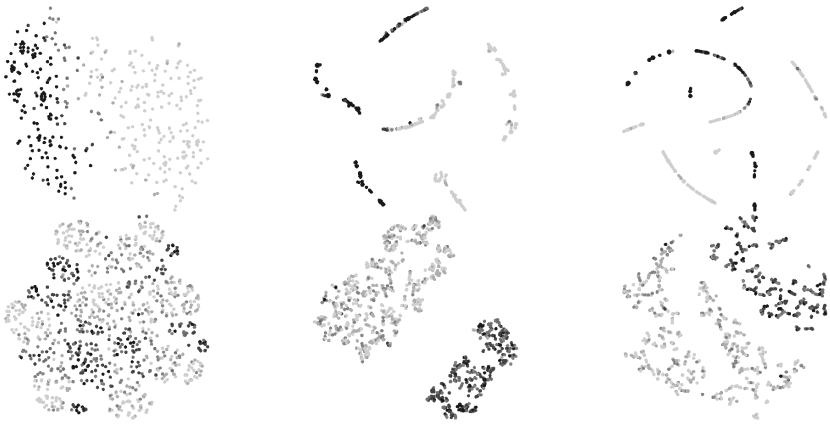


Figure 3.3. Examples of t-SNE visualizations of the raw data (left), MLPfeat (middle) and MLPfisher10 (right), for USvotes (top row) and TicTacToe (bottom row) data sets. The shade shows probability of class 1, as predicted by the model.

heuristical but much faster MLPfeat often gives equally good and sometimes better results advocates its use for model visualization, especially for larger data sets for which MLPfisher computations can become prohibitively slow.

3.4 Conclusions

Changing the metric that a DR method uses provides an easy means for adding supervisory information or information from a higher-level analysis into an unsupervised DR method. Cophenetic distances from hierarchical clustering and Euclidean metric in the feature space of an MLP model are examples of such metrics.

Combining cophenetic distances with Sammon mapping gives a space-efficient way for visualizing hierarchies. The experiment also draws new attention to Sammon mapping as a visualization tool. Sammon mapping and other distance-based DR methods have fallen in the shadow of methods that emphasize close neighborhoods. They may find a new use in situations, where the metric is used to describe the content we want to show, like when the cophenetic distance is used to encode the clustering results. In order to show the intended content, the distances in the visualization should reflect the metric. In such cases, distance-preserving methods may work better than methods that concentre on neighborhoods.

Using features from a supervised model in an unsupervised DR method results in a form of supervised DR. Many, if not most, DR methods allow using different metrics, so the choice of DR methods poses few limitations for this approach. Choice of the model is more restricted, since only models which provide an internal representation of data can be exploited. In our experiments, the simple metric built on the MLP model feature space performed as well as the theoretically better-motivated but computationally more demanding learning metric.

Modular approach that separates model and DR provides more flexibility than conventional supervised DR. Building models on data has been extensively studied, and consequently several useful tools have been developed for the task. One can choose model complexity, use prior information, and validate models. In comparison, the cost functions of supervised

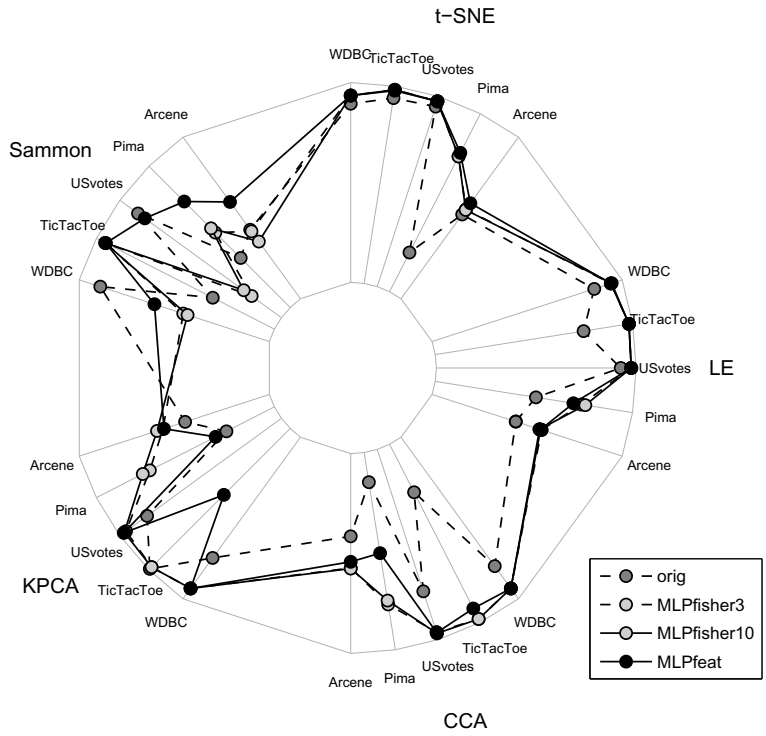


Figure 3.4. Comparison of MLPfeat and MLPfisher in different DR methods and on different data sets. Distance from the center gives the classification accuracy, 100 % at the outer circle and 50 % at the inner circle. Label "orig" refers to raw data.

DR mostly use the idea of nearest neighbor classification, possibly offering a choice of how many neighbors to use. More advanced tuning of such methods can be difficult or impossible.

4. Feature space of the Extreme Learning Machine

Dimension reduction deals with the problem of mapping data from a high-dimensional to a low-dimensional space, using a mapping with some desirable properties. Mappings between spaces are, of course, not unique to dimension reduction. A similar setting is met e.g. kernel classifiers: before the actual classification is carried out, the data is mapped to a feature space. Criteria which are used to study dimension reduction results can be used to examine and compare feature space mappings as well, and DR visualizations can be used to give a concrete idea of how the feature space transformation changes data.

In this chapter we take a look at some models where feature spaces are an essential part: MLP, two kernels commonly used in kernel classifiers (squared exponential (SE) and neural network kernel (NNK)), and Extreme Learning Machine (ELM), a neural network architecture based on random projections.

Extreme Learning Machine is our main target of study. It is a relatively new method which has been developed into many different variants, but so far its properties or connections to other models have not received much attention. ELM consists of two layers. The first uses randomly chosen weights, and the last layer learns a linear model on the features provided by the first layer. This makes ELM fast and simple to train compared to other neural network methods.

We compare the feature space of ELM to those of other models (Sec. 4.2), noticing that although ELM is a supervised neural method, its random, unsupervised feature space mapping clearly differs from the mappings learned by MLP networks. Instead, ELM features resemble those of kernel methods, which also rely on unsupervised features.

We point out a close connection between ELM and the neural network kernel. NNK is a kernel which is derived as the feature space covariance of an infinite neural network. Making the network infinite makes individual network weights meaningless, thereby resulting in features which only depend on the data points and the variance of weights. We propose a novel interpretation of ELM (Sec. 4.3) : it can be seen as an approximation of the infinite network used in the derivation of NNK.

The connection between NNK and ELM highlights a need to reassess the way ELM is usually parameterized. Currently, the only parameter considered is the number of hidden units, and ELM is promoted for its relatively easy model selection, since only one parameter needs to be chosen. At the same time, the only parameter of the related NNK is weight variance. We show (Sec. 4.4) that weight variance can have an impact on ELM results as well, and explain the role of variance in determining the properties of the feature space mapping. We also discuss the implications

of our findings on some claims made about ELM (Sec. 4.5).

The material in Secs. 4.3–4.5 has been published earlier in [134]. Sec. 4.2 is new material.

4.1 Random and infinite neural networks

In this section we describe the most relevant methods we need in this chapter: Extreme learning machine and neural network kernel. Description of MLP models can be found in Chapter 3, and squared exponential kernel will be briefly explained where it is first used.

4.1.1 Extreme Learning Machine

Extreme Learning Machine [83, 82] (ELM) is a recently proposed neural network architecture based on random projections. ELM has one nonlinear hidden layer with random weights, and an output layer whose weights are determined analytically.

ELM formalism places few restrictions to the activation of the hidden units. We use the error function $\mathrm{Erf}(z) = 2/\sqrt{\pi} \int_0^z \exp -t^2$ in all our experiments, since it is the sigmoid used in the derivation of NNK. For the same reason we use Gaussian distribution for weights, instead of the more usual uniform. ELM only requires the distribution to be continuous.

Output weights β are obtained from a linear regression model

$$\mathbf{Y} = \beta_0 + \sum_{h=1}^{H} \beta_h \mathrm{hid_h}(\mathbf{x}) \tag{4.1}$$

(where β_0 is bias term and β_h is for hidden unit h) fitted to the hidden layer outputs $\mathrm{hid_h}(\mathbf{x})$. In matrix notation, this becomes

$$\mathbf{Y} = \mathbf{H}\beta \tag{4.2}$$

$$\beta = \mathbf{H}^{\dagger}\mathbf{Y}, \tag{4.3}$$

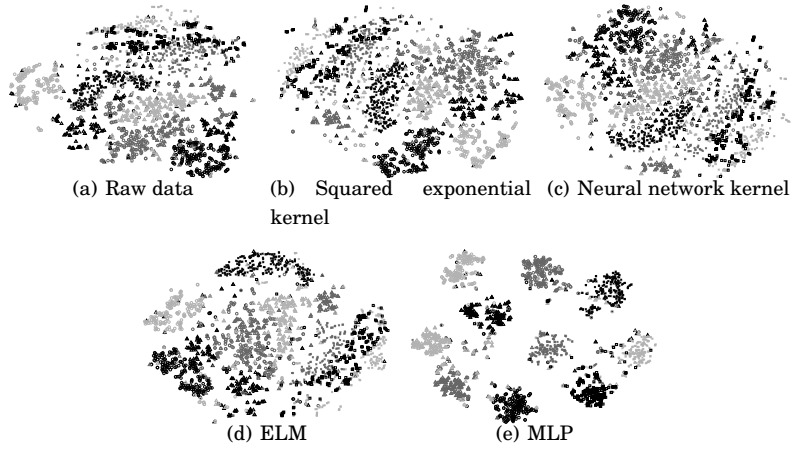
with the hidden layer outputs collected into $\mathbf{H}$ and the linear model fit realized by taking pseudoinverse $\mathbf{H}^{\dagger}$.

In regression, prediction happens by feeding the test data through the hidden layer and then using the regression weights to compute the model output. In classification, regression predictor is turned into a class number by thresholding at zero. Nothing would prevent doing classification by fitting a generalized linear model [120], which would yield class probabilities instead of a binary choice. This is however usually not done in ELM classifiers.

Basic ELM only learns the output weights, and the weights of the hidden layer remain independent of each other. Also many variants of ELM have been proposed. They effectively introduce some form of indirect training of the first layer weights. After applying the method the weights are not independent any more, but have acquired some dependencies encoding information about the training data. Dependencies are introduced e.g. by means of selecting the neurons [122] or whole networks [218] based on their ability to predict the target variable. We concentrate on basic ELM here.

4.1.2 Neural network kernel

Neural network kernel (NNK) is derived in [205] by letting the number of hidden units in a one-hidden-layer network go to infinity. A Gaus-



(a) Raw data (b) Squared exponential kernel (c) Neural network kernel (d) ELM (e) MLP

Figure 4.1. Visualization of 2000 MNIST points, mapped to different feature spaces before applying t-SNE.

sian prior is set to hidden layer weights, which are then integrated out. The only parameters remaining after the integration are variances for weights. This leads to an analytical expression for expected covariance between two feature space vectors,

$$\mathrm{k}_{\mathrm{NN}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{2}{\pi} \sin^{-1} \frac{2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_j}{\sqrt{(1 + 2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_i)(1 + 2\tilde{\mathbf{x}}_j^T \Sigma \tilde{\mathbf{x}}_j)}}. \tag{4.4}$$

Above, $\tilde{\mathbf{x}}_i = [1\ \mathbf{x}_i]$ is an augmented input vector and Σ is a diagonal matrix with variances of inputs. As NNK is derived as a covariance, it is always positive semidefinite.

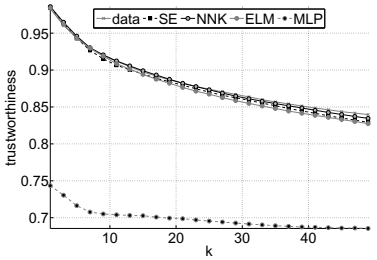
NNK also arises as a special case of a more general arc-cosine kernel [30]. Neural network kernel should not be confused with so called MLP kernel, $\tanh(a(\mathbf{x}_i^T \mathbf{x}_j) + b)$, more often known as $\tanh$-kernel or sigmoid kernel (analyzed in some detail in [114]). A classifier using this kernel implements a kind of neural network, where training data points take the role of input weights [190].

4.2 Comparison of feature spaces

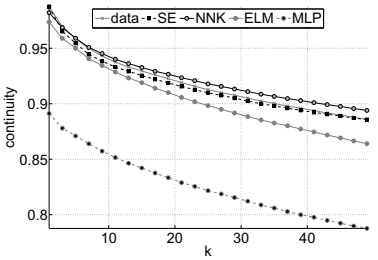
We use t-SNE to visualize feature spaces of ELM, MLP and two kernels: NNK and squared exponential. Squared exponential is a very commonly used kernel, and goes under different names, e.g. Gaussian and radial basis function kernel. It is a stationary kernel, meaning that similarity of two points only depends on their distance, and not on point locations. Similarity is given by an unnormalized Gaussian, $\exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|/\sigma^2)$.

ELM features are used like the MLP feature space in Sec. 3.1.2, just computing Euclidean distances between the feature vectors. The kernels are used to determine similarities between data points, and these are converted into dissimilarities using Eq. (2.3). Pairwise dissimilarities are then used in t-SNE instead of the usual Euclidean distance.

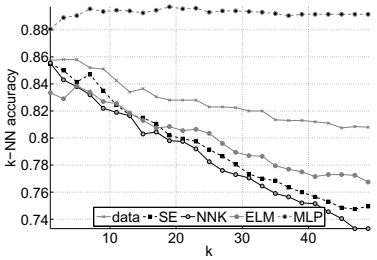
In Fig. 4.1, the visualizations obtained using kernels and ELM resemble those created using raw data. They more or less preserve the original structure of the data, as a stark contrast to the MLP feature space visualization, which clearly separates the different digit classes. This finding



(a) Trustworthiness



(b) Continuity



(c) K-NN classification accuracy

Figure 4.2. Numerical quality of visualizations in Fig. 4.1.

supports the view, presented in [206], that the choice of kernel may not matter very much, as the results depend more on the distribution of data. Here any essential differences are between unsupervised and supervised features.

SE kernel is stationary and only reacts to point distances. Although NNK uses a covariance matrix, it is always diagonal, so also NNK only considers importance (variance) of individual coordinates, ignoring their mutual dependencies.

Same holds for ELM; although neural network weights could produce location-specific behaviors, the random weights do not contain such information. When point locations are not considered, all points behave more or less the same under the feature space mapping. Especially, systematic differences between the points of different classes cannot emerge, as this would require awareness of point locations in space.

In a trained MLP network, a mapping is learned such that points of a class are kept together, and points of different classes are pushed away from each other – that is, any large nonlinearities introduced do not break random neighborhoods, but mainly occur between the classes. This behavior results in a very different feature space from the unsupervised methods, as was seen in Fig. 4.1.

Fig. 4.2 shows trustworthiness, continuity and k-NN accuracy for visualizations in Fig. 4.1, comparing the 2D results to the original data. The difference between the unsupervised and the supervised results is obvious also here. It is more interesting to notice that both trustworthiness and continuity are high for the unsupervised feature spaces.

High continuity of unsupervised mappings is not surprising, since the mappings to feature spaces are continuous. Points originally close to each other tend to remain so, more clearly for closer points. Parameters of the mapping determine how strong this behavior is. Smaller kernel widths and larger weight variances allow more drastic folding of space in the mapping, and may lead to stretching of some neighborhoods.

Trustworthiness, which suffers if two originally distant points are placed close together, is also high for kernel and ELM mappings. A kernel mapping emphasizes clusters of data by creating a matrix with large values for similar and small values for dissimilar points. For such covariance to appear, also the corresponding implicitly defined feature vectors must not only keep neighbors together, but must also have dissimilar points, i.e. non-neighbors, in different regions of feature space. This results in trustworthy mappings.

High trustworthiness of random ELM features may seem counterintuitive, since there is no explicit mechanism preventing distant points from meeting. Indeed, random mapping can sometimes produce overlapping neighborhoods, but this happens with small probability, since the feature space is fairly high-dimensional.

Again, the supervised MLP behaves differently, producing features much less true to the original neighborhood structure. Same-class points being kept together sometimes requires points from outskirts of the class to move close to points which belong to the same class but are far away distant-wise. This gives good k-NN-classification, as seen in Fig. 4.2, but less trustworthy mappings.

Kernel classifiers first map the data to the high-dimensional feature space of a kernel, and then classify. High trustworthiness and continuity of kernel mappings, combined with easier classification in high-dimensional spaces, can produce good classification results in an unsupervised feature space.

In dimension reduction, on the other hand, first mapping the data into an unsupervised feature space and then to a lower-dimensional space does not seem to bring obvious benefits. The structure of data is preserved in the possibly high-dimensional feature space, but due to high dimensionality, it may be even more difficult to find than in the original data. This casts some doubt on feasibility of methods, like kernel PCA and kernelized variants of unsupervised DR methods, where the mapping is not data-based at all. Methods which actively build a kernel to describe the metric structure of the data, e.g. by following manifolds or recognizing directions relevant to target prediction, seem better motivated from this perspective.

4.3 Extreme learning machine approximates the neural network kernel

Above we saw that the ELM feature space behaves much like that of a kernel, although ELM is predominantly known as a neural network method. In this and following sections we study the connection between ELM and neural network kernel more closely. We interpret ELM features as an approximation to NNK feature space. A similar connection probably exists between random weight radial basis function networks and squared exponential kernels, although we do not study it here.

Essential property of a fully trained neural network is its ability to extract features from the data. The features should be good for predicting the target variable of a classification/regression task. In a network with one hidden and one output layer, the hidden layer learns the features, while the output layer learns a linear mapping. We can think of this as first non-linearly mapping the data into a feature space and then performing a linear regression/classification in that space.

ELM has no feature learning ability. It projects the input data into whatever feature space the randomly chosen weights happen to specify, and learns a linear mapping in that space. Parameters affecting the feature space representation of a data point are type and number of neurons, and the variance of the hidden layer weights. Training data can affect these parameters through model selection, but not directly through any training procedure.

This is similar to what a kernel classifier, e.g. a support vector machine (SVM) [33], does. A feature space representation for a data point is derived, using a kernel function with a few parameters, which are typically chosen by some model selection routine. Features are not learned from data, but dictated by the kernel. Weights for linear classification or regression are then learned in the feature space. The biggest difference is that where ELM explicitly generates the feature space vectors, in SVM or another kernel method only similarities between feature space vectors are used.

The mapping ELM uses is random. Therefore, the individual weights of the ELM hidden layer have little meaning, and essential information about the weights is captured by their variance. This thought is similar to derivation of the neural network kernel, which is parameterized with weight variance.

We interpret ELM as an approximation to the infinite neural network, which is used to derive NNK. In the experiments below, we demonstrate that ELM and NNK can, to certain extent, replace each other in computations. Namely, ELM can be used to compute a kernel, whose behavior approaches that of NNK when the number of hidden units grows. On the other hand, decomposition of NNK produces features which can be used instead of the random features of ELM computations.

We use five binary classification data sets, detailed in Appendix A.

4.3.1 ELM kernel

We can use ELM hidden layer to compute a kernel, and use it in any kernel method. This idea has been suggested for support vector machine in [57] and briefly tried in Gaussian process classification in [134].

ELM kernel function [57] is defined as

$$\mathrm{k}_{\mathrm{ELM}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{H} f(\mathbf{x}_i) \cdot f(\mathbf{x}_j), \tag{4.5}$$

that is, the data is fed trough the ELM hidden layer to obtain the feature space vectors, and their covariance is then computed and scaled by the number of hidden units.

When the number of hidden units grows, this kernel matrix approaches that given by NNK. Fig. 4.3 shows the approach of ELM to NNK, measured by the Frobenius norm, as function of H. Although some variance due to random weights remains, ELM kernel seems to converge towards NNK.

4.3.2 Deriving ELM features from NNK

We can use NNK to replace the hidden layer computations in ELM. This is done by first computing a similarity-based representation for data points using NNK, and then deriving a possible set of explicit feature space vectors by matrix decomposition. This corresponds to using ELM with an infinite number of hidden units.

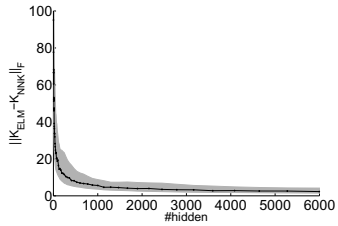


Figure 4.3. ELM kernel (mean by black dots, 95% interval by shading) approaches the neural network kernel in Frobenius norm, as function of H. Variation is caused by randomness in weights. WDBC data set was used.

When using ELM, we only deal with vectorial data, with data space vectors transformed into feature space vectors by the hidden layer. Kernel methods rely on pairwise data, where only similarities from any point to all training points are considered. Kernel matrix specifies the pairwise similarities. In order to use pairwise information from the NNK instead of ELM hidden layer, we must find a vectorial representation for the data.

As a covariance matrix, NNK is positive semidefinite. Any PSD matrix can be decomposed into a matrix and its Hermitian conjugate

$$\mathbf{C} = \mathbf{L}\mathbf{L}^H. \tag{4.6}$$

There are different methods for finding the factors [67]. Matlab `cholcov` implements a method based on eigendecomposition. If we take $\mathbf{C}$ in Eq. (4.6) to be output of the NNK function Eq. (4.4), then $\mathbf{L}$ can be thought as one possible set of corresponding feature space vectors.

We use $\mathbf{L}$ to determine the output layer weights the same way we used $\mathbf{H}$ in ELM,

$$\beta = \mathbf{L}^\dagger \mathbf{Y}. \tag{4.7}$$

The factors $\mathbf{L}$ are unique only up to a unitary transformation, but this is not a problem in ELM context, as the linear fitting of output weights is able to adapt to linear transformations.

With an infinite number of hidden units, the feature space is infinite-dimensional. Meanwhile, the data we have available is finite, and the N data points span at most an N-dimensional subspace of the infinite-dimensional feature space. Therefore, the size of $\mathbf{L}$ is at most $N \times N$; the number of columns can be smaller.

The one remaining problem is the mapping of test points to the feature space. In ELM, the test data is simply fed through the hidden layer. In our case, the hidden layer does not physically exist, and we must base the calculations on similarities from test points to training points, as given by NNK Eq. (4.4). This means that NNK output for test data $\mathbf{C}_*$ is a covariance matrix of the form

$$\mathbf{C}_* = \mathbf{L}\mathbf{L}_*^H. \tag{4.8}$$

We have already determined the pseudoinverse of $\mathbf{L}$when training the model with Eq. (4.7). $\mathbf{L}_*$ is recovered from

$$\mathbf{L}_* = (\mathbf{L}^\dagger \mathbf{C}_*)^H = (\mathbf{L}^\dagger \mathbf{L}\mathbf{L}_*^H)^H, \tag{4.9}$$

and the predictions for test targets are computed as

$$\mathbf{Y}_* = \mathbf{L}_* \beta. \tag{4.10}$$

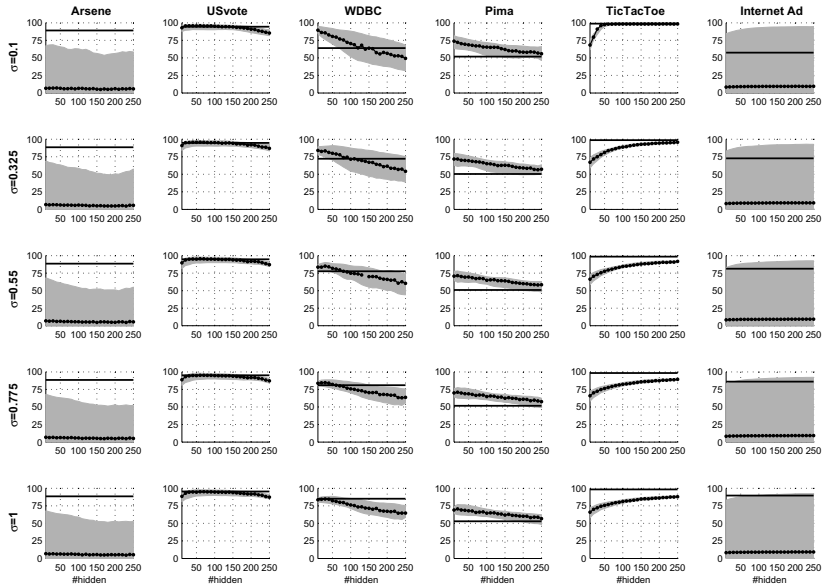


Figure 4.4. ELM results (mean as black dots, 95 % interval as shading) for different values of σ. Variation is that due to data, results have been averaged over 10 ELM runs to handle uncertainty from random weights. The means of NNK results (horizontal line) are shown for comparison.

In Fig. 4.4 we compare the predictions given by ordinary ELM to those produced using feature vectors derived from NNK. We notice that when the variance is properly chosen, using NNK gives equal or better results than ELM for most data sets. Pima data set is an exception, ELM has some predictive power whereas NNK-based network performs almost at level of guessing.

We also notice that the choice of variance has a marked effect on two and some effect on other data sets, both for ordinary ELM and the NNK variant. In the following section we will look at variance effects in more detail and discuss the reasons for importance of variance.

4.4 Variance parameter in ELM

An infinite network performing equally well or often better than ELM raises a question about meaningfulness of choosing model complexity based on hidden units only, as is traditionally done with ELM. NNK is parameterized using weight variance. In this section we study effect of variance on ELM, concluding that variance should be used as a tuning parameter in ELM as well as in NNK.

4.4.1 Experiments

For simplicity, we assume all variances equal both in NNK and ELM. It would also be possible to use a separate variance parameter for each input variable.

Variance effects already noticed in Fig. 4.4 are summarized in Fig. 4.5. Averages over repeated data samples are shown. Maximum number of hidden units for ELM experiment is 250, to make sure to stay in the

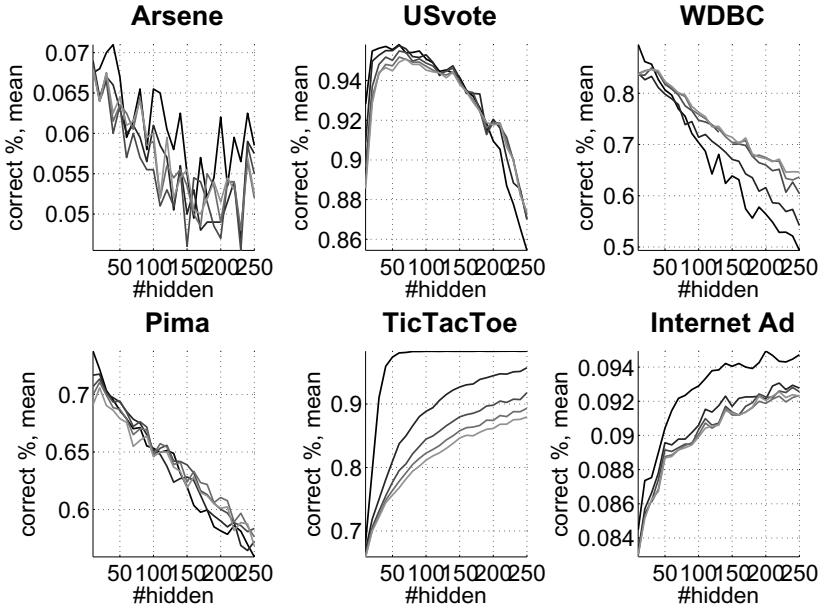


Figure 4.5. Effect of variance on mean of ELM predictions. Darker shade indicates smaller variance.

sensible operating range of ELM (up to N hidden units) for all data sets, some of which are small.

Fig. 4.5 shows the mean predictions of ELM as function of H, for various values of variance. For TicTacToe and WDBC data sets the predictions are clearly affected by the variance parameter. For Internet ad data the overall effect of both H and σ is very small. In that scale, the smallest variance nonetheless gives clearly different results than the larger values. Results for other data sets are not very sensitive to the variance values that were tried. For TicTacToe and Internet ads smaller variance gives better predictions, for WDBC the biggest one does. Clearly no fixed variance can be used for all data sets.

4.4.2 Explaining the results

When thinking about the mechanism by which the variance parameter affects the results, differences between data sets are to be expected. Variance affects model complexity, and obviously, different models fit different data sets. Variance and distribution of the data together determine the magnitude of values seen by a hidden unit. The operating point of the sigmoidal unit determines the flexibility of the model. When weights are small, the sigmoid produces a nearly linear mapping. Large weights result in a highly non-linear mapping. This is illustrated in Fig. 4.6. One-dimensional data points, spread over range [-1,1] (the x-axis), are given random weights drawn from a zero-mean Gaussian distribution and then fed through an error function sigmoid, repeating this 10000 times. Mean output and 95 % interval are depicted. On average, the sigmoid produces a zero response, but the distribution of responses is determined by the variance used. Small variance means mostly small weights, and linear operation. Large variance produces many large weights, which increase the proportion of large responses by the network, allowing nonlinear map-

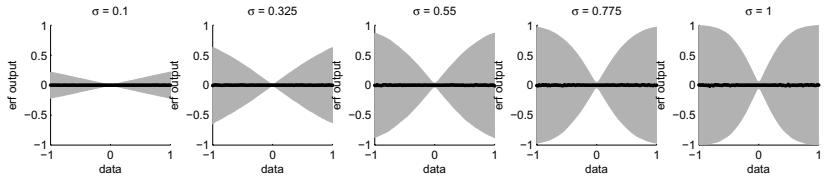


Figure 4.6. Distributions of predictions of an error function sigmoid for different σ.

pings.

4.5 On properties of ELM

Authors of [83] promote ELM by speed, dependence on a single tuning parameter, small training error and good generalization performance. These claims have often been repeated by subsequent authors, but we have not come upon much discussion of them. Here we present some comments on these properties, largely based on our observations about the role of weight variance.

Training of a single ELM network is fast, provided the number of hidden units is small. *Speed of training* as the whole, however, depends also on the number of individual training runs. Model selection may require considerable number of repetitions.

Complexity of model selection is determined by the *number of tuning parameters*, since all sensible value combinations should be considered. First parameter is the number of hidden units. The only theoretically motivated upper limit for the number of hidden units to try is N (which is enough for zero training error). At that limit, computing pseudoinverse corresponds to ordinary inversion of an $N \times N$ matrix, with a complexity of $O(N^3)$. In practice, smaller upper limits are used.

Traditionally, somewhat arbitrary fixed values have been used for weight variance, and model selection has only considered the number of hidden units. Importance of variance, or more often the range used for uniform distribution, is recognized in ELM works (e.g. [122]). However, it is not seen as a model parameter, but simply a constant which must be suitably fixed to guarantee that the sigmoid operation neither remains linear nor too strongly saturates to ± 1.

Our results show that also the weight variance has an effect on results, and should thus be considered a tuning parameter. Although both the number of hidden units and the variance affect flexibility of mappings the network can implement, variance is more clearly associated with model linearity or nonlinearity. For neural networks with non-random hidden layers, weight variance is known to have more effect than network size [5]. Perhaps variance should be the primary tuning parameter for ELM as well.

Generally, *small training error* and good generalization may be contradictory goals. ELM is proved [83] to be able to perfectly classify its training data, if the number of hidden units equals or exceeds the number of data points. This behavior, though important in proving computational power of ELM, is usually not desirable in modeling. A model should generalize, not exactly memorize the training data. This view is indirectly acknowledged in practical ELM use, where the number of hidden units is much smaller than N. This may prevent ELM network from overfitting

to the training data, a factor usually not discussed in ELM literature.

Generalization ability of ELM is attributed to the fact that computing output layer weights by pseudoinverse achieves a minimum norm least-squares solution. The generalization ability of a neural network is in [5] shown to relate to small norm of weights. However, [5] considers the neural network as whole, not only the output layer. Although ELM minimizes the norm of the output layer weights, the norm of the hidden layer weights depends on the variance parameter, and does not change in ELM training.

In the hidden layer, the generalization ability is related to the operating point of hidden unit activations, discussed in Sec. 4.4. A model with small hidden layer weights is nearly linear, and generalizes well. A highly non-linear model, produced by large weights, is more prone to overfitting. Therefore, conclusions about the generalization ability of ELM should not be based on the output weights only.

4.6 Conclusions

In this chapter we studied ELM, a neural network method whose functionality in some sense falls between a neural network and a kernel method. ELM classifies data based on random features. DR visualizations of feature spaces of ELM, a trained MLP, and two different kernels illustrate the resemblance of the ELM feature space to kernel feature spaces. Both kinds of feature spaces seem to preserve the structure of the original data fairly closely, strengthening a finding by [206] that choosing a kernel may not matter very much, since the results mostly depend on data.

We gave a new interpretation of ELM as an approximation of an infinite neural network, such as is used in the derivation of the neural network kernel. The close connection between ELM and NNK was demonstrated by showing that they can, to some extent at least, replace each other in computations.

In NNK, the only parameter is weight variance. This is in stark contrast with ELM, which is usually only parameterized by the number of hidden units, while weight variance is arbitrarily fixed. We studied the effect of weight variance in ELM, and found it to affect the results. Although the role of variance is known in other neural network research, to the author's knowledge its importance has not been pointed out in ELM context before. The finding has practical implications for how ELM models should be constructed. Especially model selection, if done carefully, becomes more time-consuming than earlier ELM works claim, since two parameters must be considered.

5. Supervised dimension reduction with bottleneck networks

Bottlenecked regression and classification networks can be thought as supervised versions of autoencoders. Their idea is very simple: the network has a low-dimensional bottleneck layer like autoencoders have, but the network is trained to perform nonlinear classification or regression, not to reconstruct the data.

One of the current topics in neural network research are deep networks. Their training remained a challenge until 2006, when a method of unsupervised pretraining was developed [81]. Pretraining finds a good initialization for weights, so that good solutions can be found by gradient descent methods. The reasons for why and how pretraining works are not completely clear [47]. In spite of this, the practical possibility to train deep networks has in practice started a new branch of neural network research, including alternative pretraining methods [192], studies about pretraining effects, [47], criticism of methods [51], and use in dimension reduction [151, 123, 180, 124], among other applications.

Some work on neural networks and dimension reduction, notably on autoencoders but also other methods, was done in 1990s with shallow networks. The surge of interest to deep networks has meant revival or re-invention of ideas for doing dimension reduction with neural networks. Unsupervised dimension reduction with deep autoencoders was followed by supervised networks with DR-oriented, quadratically scaling cost functions [151, 123], which combine deep nets and supervised DR exactly the same way as shallow networks and Sammon mapping were combined in [118].

Although autoencoders have long been a familiar tool for dimension reduction, supervised dimension reduction was not a topical task in 1990s, and the supervised variant of bottleneck networks that was developed at that time [84] did not find widespread use. When the idea of using deep networks for supervised dimension reduction came up, supervision was done by quadratically scaling cost functions. This development leaves a void, which inevitably calls for re-introduction of the simpler way of supervision. In this work we fill this void by a deep-net variant of supervised bottleneck classifiers (Sec. 5.3). Our work contributes to the current deep network research by showing that supervised bottleneck networks can perform as well as the currently used supervised neural DR methods, while being faster to train. The bottleneck classifier experiments have been published in [130].

Although from neural network point of view classification and regression are closely related tasks, in the field of supervised dimension reduction these two problems are usually solved with different methods. Dimension reduction for regression (DRR) is a research branch of its

own. With its roots in multivariate statistics, it has developed somewhat separated from the more clearly machine learning oriented methods, which mostly use class labels for supervision. We therefore complement our experiments on the bottleneck classifiers with a separate study (Sec. 5.4), which compares bottleneck regression networks to established DRR methods. Unlike the classifier case, we use shallow networks, since the comparison methods can only be used on smallish data sets, not sufficient for the data-intensive pretraining of a deep network. We are not aware of earlier use of bottleneck networks in DRR, although related network architectures have been applied in other fields. Part of the DRR experiments is from [131], while most is new material.

5.1 Feedforward neural networks in dimension reduction

5.1.1 Autoencoders and their variants

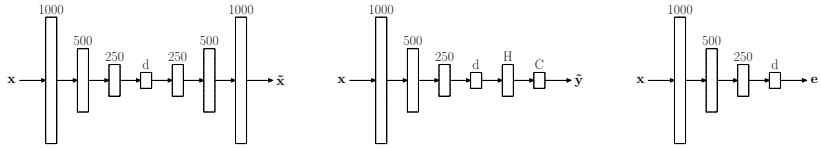
Autoencoders were proposed as a nonlinear extension of principal components analysis (PCA) already two decades ago [46, 155, 97, 41]. An autoencoder is a feedforward neural network with (usually) symmetric layer structure and a low-dimensional middle layer (see Fig. 5.1(a) for illustration). It is trained to reconstruct the data, by presenting the same data to the network both as the input and as the learning target. Dimension reduction happens at the middle layer, whose output provides a low-dimensional representation of the data. The connection to PCA stems from the observation that a network with autoencoder structure, when using linear hidden units, learns the highest principal components of the data [4]. Not until in 2000 it was fully recognized that using non-linear hidden units leads to a non-linear model [89]. Nowadays, and also in this work, the term autoencoder usually refers to a network with sigmoidal hidden units, with linear units in the middle and output layers only.

Although the idea of autoencoders is old, until now they were little used due to problems of training. With invention of efficient pretraining methods, the deep networks, among them autoencoders, have become a current topic.

Several ways of introducing some supervision into autoencoders have been proposed. In divergent autoencoders [99], the layers up to the bottleneck are shared by all classes. After the bottleneck, the network diverges into several parts, each of which is trained with samples of one class only. Supervisory information can also be added via regularizers [204], either layerwise or to the cost function of the whole network. A resembling approach is to couple each layer with a linear classifier, and use a tuning parameter to decide the respective importance of unsupervised and supervised costs in training [145].

5.1.2 Pretraining deep networks with a generative model

Feedforward neural networks are usually trained using gradient descent algorithms. The gradients are computed by backpropagation, invented already in the 1970s but made widely known in 1986 [150]. The gradients of the network error w.r.t. layer weights are obtained by chain rule (layer error w.r.t. layer output, layer output w.r.t. layer weights). The key observation is that the current-layer error can be computed from the next layer gradient, using network weights to determine respective contributions of the current layer units in the formation of error. Using this



(a) Autoencoder.

(b) Bottleneck network (H hidden units, C-dimensional output).

(c) Quadratic cost functions.

Figure 5.1. Illustration of network layouts for the bottleneck network and the comparison models, each performing dimension reduction to d dimensions. The d-dimensional coordinates are the outputs of the middle layer (for the bottleneck networks) or the output layer (with the quadratic cost functions). The autoencoder (a) minimizes mean square reconstruction error to achieve reconstruction $\tilde{x} \approx x$ when trained with input x and target x. Error function of the supervised bottleneck network (b) is cross-entropy classification error for the classifier, and mean square error for the regression networks. The network produces an estimate $\tilde{y} \approx y$ when trained with input x and target y (y are either class numbers, presented in 1-of-c binary encoding, or regression targets). With the quadratic cost functions (c), distance between two low-dimensional points, obtained from the network output e is compared to the distance between the corresponding input points, using a suitable cost function (see text).

observation, the gradients can be computed layer by layer, starting from the last and backpropagating the error to previous layers.

Deep neural networks can learn mappings more effectively, that is, with less neurons, than shallow networks [13, 10]. As a downside, complicated energy landscapes of deep networks, are a challenge to gradient descent algorithms. Training is likely to land in a local energy minimum, and can result in a badly trained network. Due to the huge number of possible weight combinations, looking for a global optimum is not a feasible strategy. Instead, better solutions can be looked for by choosing the initial values of weights so that the starting point for gradient descent is near a good optimum.

The currently used way of training deep networks follows a two-phase paradigm, started by the breakthrough work of [81].

First, the network is pretrained in unsupervised fashion, even if the final use of the network will be in a supervised task. Then it is finetuned by minimizing the network cost function, which trains the network e.g. as an autoencoder or a classifier. Since pretraining has found good initial values for weights, finetuning can be done by gradient descent methods [47].

Currently, two main strategies for pretraining deep networks are to train each layer as a restricted Boltzmann machine (RBM) [81], or as a shallow autoencoder [12, 102, 192]. Here we use RBMs for pretraining.

A restricted Boltzmann machine [165] is a generative model of data. It consists of two layers of nodes, connected in bipartite fashion. Nodes of the visible layer present values of data, and the hidden layer nodes can be thought as latent variables generating the data. Weights between the layers are found so that the distributions of visible and hidden node values match as closely as possible, measured by Kullback-Leibler divergence. The gradient of this cost contains a term which cannot be analytically calculated. Therefore, an approximate cost function called contrastive divergence is developed in [79]. It has a simpler gradient, so that the cost

can be minimized by ordinary gradient descent.

Several RBMs can be collected into a stack, resulting in a deep generative model (deep belief network or DBN, [80]). The layers of the stack are trained one at a time, using the hidden layer output of one RBM as the visible layer input for the next one. Connection between DBNs and feedforward neural networks is made in [81], where it is noticed that a deep neural network can be initialized with DBN weights. This is a very natural idea for autoencoders, which resemble a generative model in that they attempt to reproduce the data, but the same initialization can be used for deep classifiers [102].

5.1.3 Network cost functions scaling O(N^2)

Networks with quadratically scaling cost functions have been used to create deep supervised networks for dimension reduction. Two such cost functions will serve as comparison methods in our bottleneck classifier experiments in Sec. 5.3.

Usually feedforward networks are trained by presenting the net with an input and a target, computing errors for each data point, and adjusting the weights to reduce the error.

It is also possible to train a network with a cost function which uses pairwise comparisons of network outputs to evaluate the cost. Such cost functions scale quadratically in the number of data points, so from now on we refer to them as quadratic cost functions.

A network with a quadratic cost function has no layers after the bottleneck (see Fig. 5.1(c)), but the cost is defined directly on outputs of the bottleneck layer. A separate target variable is not used. Instead, the cost function specifies requirements for point relationships, and the results of pairwise comparisons determine the error. This means that all inputs must be presented to the network at once, and that the cost of evaluating the error is quadratic in the number of training points. Any cost function which is defined for a matrix of pairwise (dis)similarities can be used.

After computing the error, the need to adjust weights is determined by backpropagating errors the same way as for usual cost functions.

The idea of using cost functions of this type is not new. In 1990s, a feedforward neural network was trained in [118, 115] to minimize the cost function of Sammon mapping [152], and was also studied in [36]. The idea of quadratic cost functions surfaced again after the invention of pretraining with generative models, now in connection with deep networks and supervised cost functions. It has been used with NCA [151] and LMNN [123] cost functions. Also a deep net version of t-SNE cost has been implemented [180], as well as a t-distributed extension of NCA [124].

5.2 Supervised bottleneck networks

Like the idea of quadratic cost functions, the idea of supervised bottleneck networks [84, 59] and related nonlinear discriminant analysis networks [117] was tried during the neural network boom of the 1990s. Supervision in a bottleneck network was noticed to make the network ignore directions that have no relation to classification [84], and the effect was also seen in visualizations. Nonlinear discriminant analysis [118] resembles the idea of bottleneck classifiers in that the network reduces dimensionality, but the network is trained as a linear classifier that gives a

low-dimensional output, not as a bottleneck network.

In DR context, the goal is to find a low-dimensional representation, and some compromizes can be made regarding to quality of predictions that can be made on DR results. The view of sacrificing some accuracy for size is especially strong in visualization tasks, where dimensionality is drastically reduced, so that some prediction ability is almost necessarily lost. In other fields of research, networks that reduce dimensionality as an intermediate step have been used for other goals than DR. Creating a bottleneck effect can force the network to generalize better, since the degrees of freedom in the bottleneck are not sufficient to repeat noise in the data. In such networks, addition of a bottleneck is expected to improve predictions. One example where such effect is sought are convolutional networks [105].

Computational resources of 1990s were usually not sufficient for handling large data sets or training deep networks with many hidden units. This work brings the idea of supervised bottleneck networks up to date, using modern pretraining and real data sets of tens of thousands of samples, and comparing bottleneck networks to recently developed dimension reduction methods.

Bottleneck classifier networks have been interpreted as generalizations of discriminant analysis. The same way as linear-unit autoencoders were shown [59] to perform PCA, bottleneck classifiers with linear units were shown to give results similar to linear discriminant analysis (LDA), and in [154], a multiclass classifier network is trained using the LDA cost function to obtain weights for the output layer, an optimization task which requires a costly eigendecomposition. The relation of the internal representation of a classifier network and discriminant analysis is also noted in [199].

Neural networks naturally bridge the gap between classification and regression methods: a multilayer network performing regression turns into a classifier by a simple change in the activation function of the output layer. Therefore, neural networks can be easily adapted to both dimension reduction for regression, and supervision with class labels. Training a bottleneck network scales linearly in the number of data points, which is a benefit over many dimension reduction methods with more complex cost functions. Also, networks are relatively insensitive to structural parameters (number of hidden units).

The appeal of neural network models in dimension reduction is largely due to their ability to map data from the input space to the low-dimensional space. Once the mapping has been learned, it stays the same, no matter whether training points or test points are fed through the network. For a neural network this behavior is obvious, but in the field of dimension reduction not so. Many methods in unsupervised dimension reduction only produce locations for the training points, with no unambiguous way of determining where the test points should go. The ability to treat the training points and the test points similarly is especially important in supervised dimension reduction, where it is natural to build models using the training data and test the prediction ability with new points.

5.3 Bottleneck classifier experiments

This chapter contains the experiments of [130].

We compare bottleneck classifiers (BC) to two closely related deep net-

work architectures: autoencoder (AE), and networks finetuned with a quadratic cost function. Two of the quadratic cost functions are supervised (NCA and LMNN), and one is unsupervised (t-SNE). These cost functions were described in Chapter 2. To justify the often heavier computation required in the supervised setting, a supervised method should produce better results than a good unsupervised one. As t-SNE has achieved remarkably good results on several data sets, especially on clustered data, we include it in our comparison as the unsupervised baseline method.

We use MNIST, USPS and 20 newsgroups data sets, and reduce the dimension to 2D, which is easy to visualize, and to 30D, which could be appropriate in a data-packing problem.

Network layouts for the methods are summarized in Fig. 5.1. A deep layout, with pretraining as an RBM stack and finetuning with the respective cost function, is used for all networks. See Appendix A for parameter values and other details.

In the autoencoder, the encoder layers (those before the bottleneck) use DBN weights as such. The decoder layers (after the bottleneck) reuse the same weights in mirrored fashion, so that the first layer after the bottleneck uses the same weights as the last layer before it, and so on. In the bottleneck classifier, the encoder layers are initialized from the RBM stack, and the classifier layers are randomly initialized.

For each data set we first pretrain a network, and use it as the starting condition for all the cost functions in turn, so that all methods start their finetuning from exactly the same configuration. The finetuning error is backpropagated through the network. A conjugate gradient optimizer is used for minimizing the error.

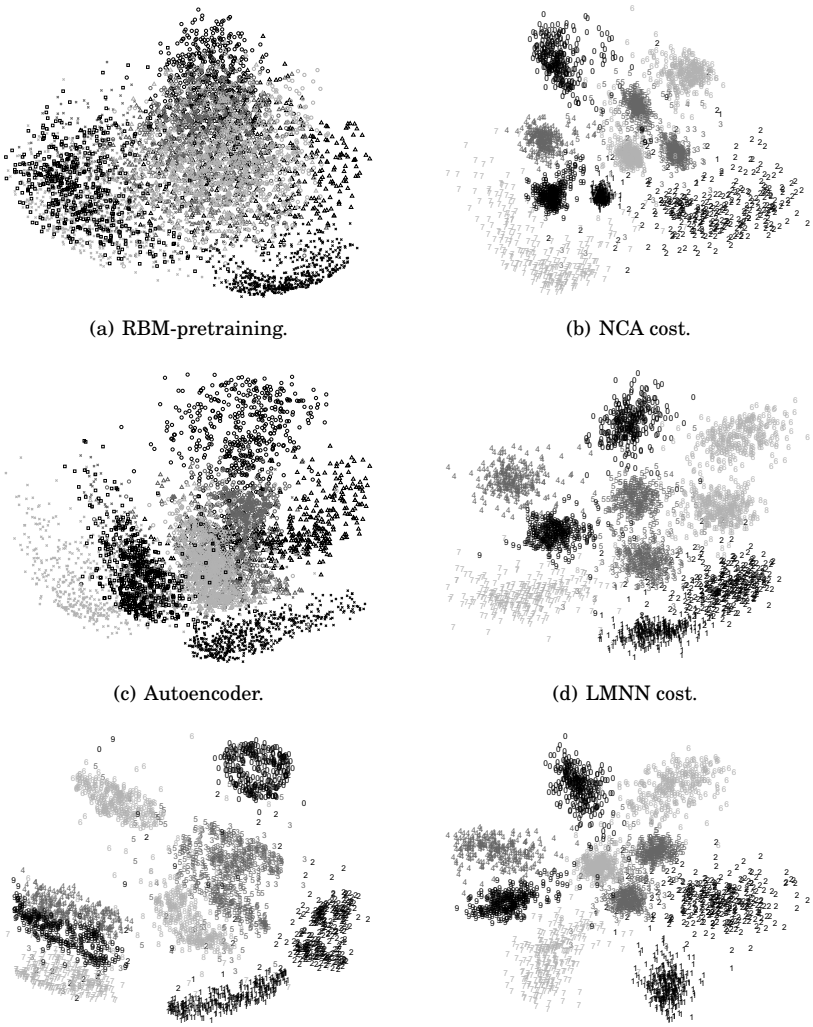
5.3.1 Results

Figs. 5.2 and 5.3 show 2D visualizations of MNIST and USPS data sets. Visualizations of the newsgroups data did not show much structure with any method, and only numerical results shown (Fig. 5.4). T-SNE does clearly better than the other two unsupervised conditions (pretrain only and the autoencoder).

For MNIST, the supervised methods separate the challenging digit groups 4–7–9 and 3–5–8 better than t-SNE does. The other digits are handled about as well by all methods. Based on visual inspection, all supervised methods seem to do equally well with MNIST data, although the numerical results differ. With the more difficult USPS data, the supervised methods keep the classes together better than t-SNE. LMNN results seem weaker than those of NCA and BC, but whether NCA or BC performs better cannot be determined on visualization only.

Numerical results for 2D and 30D cases are shown in Fig. 5.4. We use k-NN classification accuracy as our quality measure, since keeping different classes separate is a natural goal for class-supervised dimension reduction. Here we show k-NN results as function of k. Changing k does not seem to change the order of superiority of the methods (with exception of newsgroups 2D results where BC/NCA results are very close to each other). In 2D, BC does better on MNIST and USPS, and equals NCA with newsgroups data. In 30D, NCA and LMNN give lower errors.

Based on these results, no clear order of superiority can be established between the three supervised methods. Therefore, issues like computational complexity become important when choosing a method. The bottleneck classifier is worth considering because of its computational simplicity. Good performance of BC in 2D case suggests that the bottleneck



(a) RBM-pretraining.

(b) NCA cost.

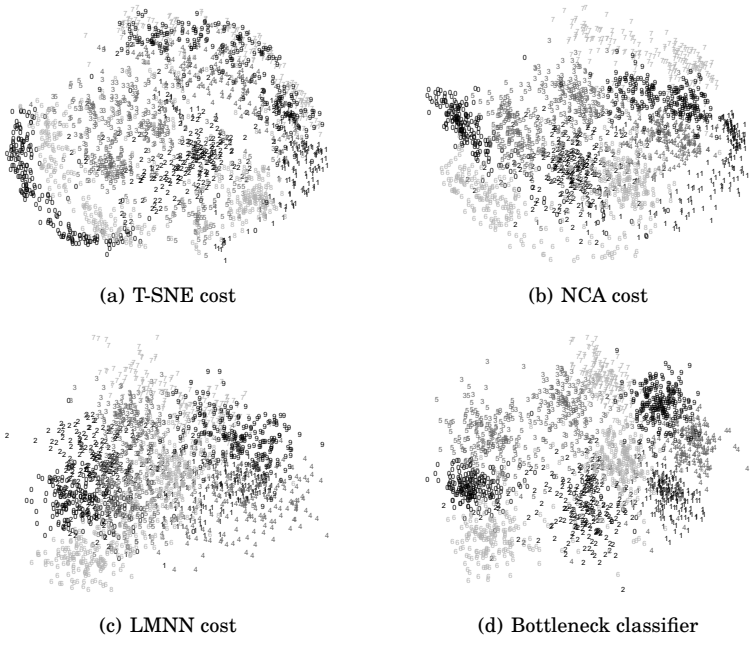
(c) Autoencoder.

(d) LMNN cost.

(e) T-SNE cost.

(f) Bottleneck classifier.

Figure 5.2. Visualizations of the MNIST test data. The left column shows unsupervised and the right column supervised results. For readability, only half of the points have been plotted. The visualization does not show clear quality differences between the different supervised methods. For numerical comparisons, see Fig. 5.4.



(a) T-SNE cost (b) NCA cost

(c) LMNN cost (d) Bottleneck classifier

Figure 5.3. Points from USPS test data in 2D. NCA and BC do better on this data than LMNN, and all supervised methods perform better than the unsupervised t-SNE.

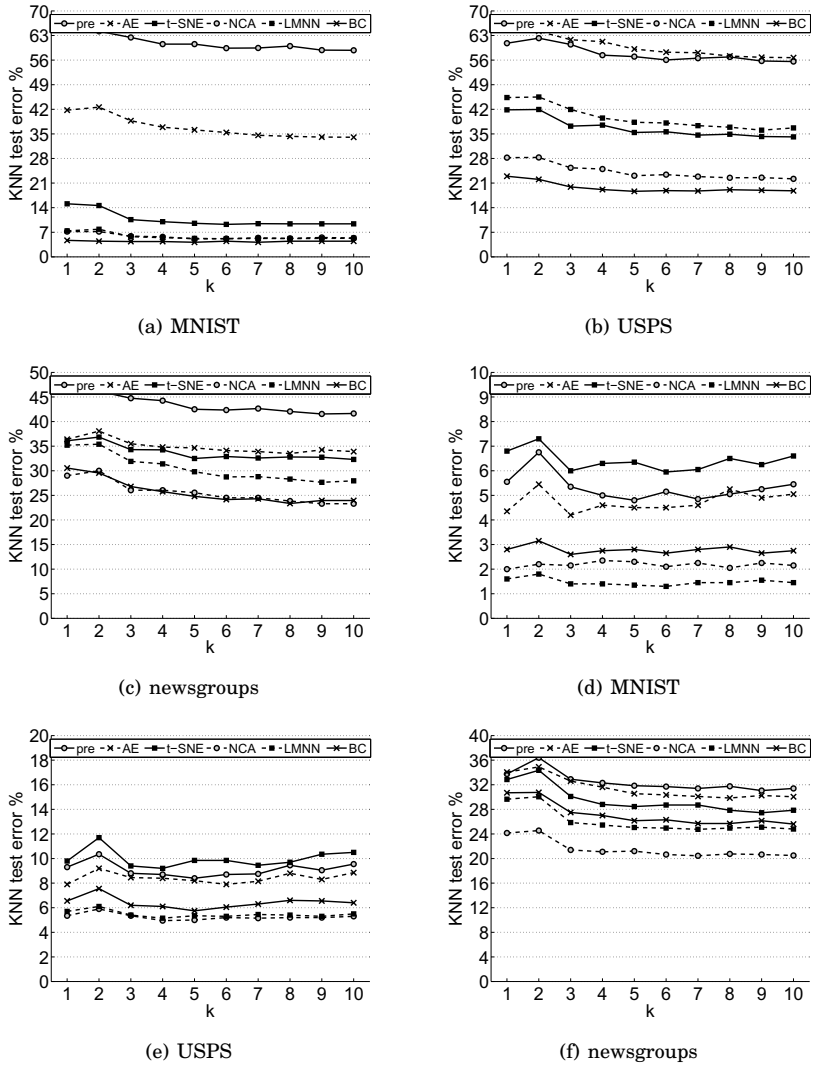
classifier could be the method of choice for visualization tasks, where the representation is forced to 2D or 3D, and NCA would do better in data-packing tasks where the embedding dimension can be higher.

5.4 Bottleneck regression network experiments

We complement the bottleneck classifier results by studying performance of supervised bottleneck regression networks (BRN) in dimension reduction for regression. We are not aware of other neural network based DRR methods, so we use other established DRR methods for comparison. This experiment is an extension of that in [131], where a comparison between BRN and covariance operator inverse regression (COIR) was performed.

We compare BRN to two linear methods (sliced inverse regression (SIR) and kernel dimension reduction (KDR)) and two nonlinear methods (manifold KDR (mKDR) and COIR). Two versions of BRN are implemented: one with a nonlinear first layer (noted simply BRN), and a linear BRN (linBRN), which is able to learn only linear subspaces and therefore comparable to linear methods. LinBRN has no layer before the bottleneck, since having a linear mapping from the inputs to the hidden units and another linear mapping from the hidden units to the bottleneck units would be superfluous; the same operation can be implemented by connecting the inputs directly to the bottleneck layer.

Two-dimensional representations of five data sets with varying input dimensionalities are sought. Data sets used are Head Pose, Yale, Parkinson, Concrete and Community crime (see Appendix A). As some of the data sets are quite small, using a deep network seems like an overkill,



(a) MNIST (b) USPS

(c) newsgroups (d) MNIST

(e) USPS (f) newsgroups

Figure 5.4. K-NN errors (for the test data) for the different data sets and methods. The top row shows 2D results, the bottom row the 30D case. As expected, the supervised methods do better than the unsupervised ones. BC works best in 2D, NCA and LMNN in 30D. Errors have been evaluated for 500 test points, finding the nearest neighbors among 2000 training points.

and there would not be enough data for RBM pretraining. We therefore use a shallow architecture, with layers numbered as L1, L2 (bottleneck), L3, and L4 (output). the network is trained using simulated annealing [157]. Since even stochastic training is affected by initial values, a multistart strategy is used (the best result of 20 runs is kept). Details of training are given in Appendix A.

Parameters for the comparison methods are chosen by trying several values and using the best results. For KDR with Yale and Parkinson data, only 1500 randomly chosen training samples are used, to keep running times reasonable. For the other methods and the other data sets, full train and test data are used.

5.4.1 Results

Visualizations of three data sets are shown in Fig. 5.5. Of nonlinear methods, COIR and BRN give good results for all data sets. Also linear methods find reasonable subspaces, but results for all of them seem worse than those of nonlinear methods. This is to be expected, since at least Yale and Head Pose data are known to be sampled from nonlinear manifolds.

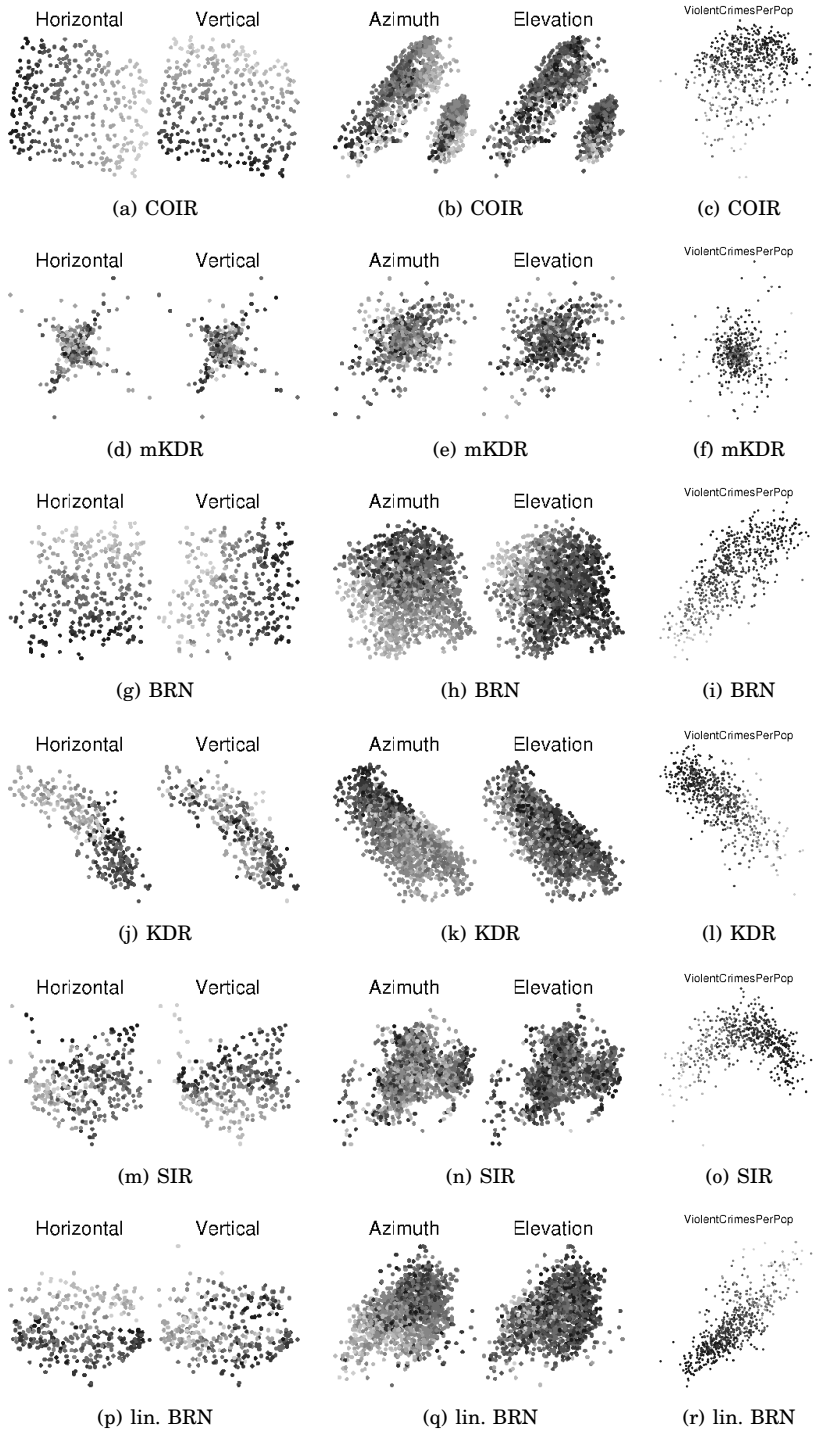
We measure numerical quality of results by nearest neighbor regression [34]. In NN-regression, y_i is predicted as the mean of the target values of the nearest (in the 2D representation) train data neighbors of a test point i. Results, as function of number of nearest neighbors (k), are shown in Fig. 5.6 for the linear methods (linear BRN, SIR and KDR) and in 5.7 for the nonlinear methods (BRN, mKDR and COIR).

Generally, both linear and nonlinear BRN do well in comparisons. With Concrete data in Fig. 5.6, it is difficult to choose any one method as best, since the results depend heavily on k. With Yale data, both SIR and linBRN perform well. With Parkinson data, linBRN fails to find a good mapping.

All linear mappings make close x-space neighbors into close z-space neighbors. Therefore NN-errors with small k do not tell very much. NN-errors for larger k tell more clearly about points with wrong y-values entering a local neighborhood. This indicates that the slope of the chosen 2D-plane is not best possible.

For most data sets, errors of the linear BRN have a fairly flat profile; error increases slowly as function of k. SIR and KDR usually show a steeper increase in errors. While BRN does not usually give the best 1-neighbor predictions, when more neighbors are used, for three data sets it gives consistently lower errors than the other methods. This suggests that linBRN finds directions that spread the data points out, so that the overall structure of data is better seen.

Similar behavior is seen in the comparison of nonlinear BRN to mKDR and COIR, Fig. 5.7. Manifold-KDR errors are small for small k, but increase rapidly. This tells about tendency of the mKDR mapping to collapse several unrelated x-points to same z-space region; nearest neighbor might be a correct one, but a bit larger neighborhood already contains a lot of alien points. This is also seen in the visualization in Fig. 5.5 where the mKDR plots, especially of Head Pose and Crime data sets, are much more crowded than the plots of BRN and COIR. COIR and BRN errors increase more slowly, indicating a good spread of points. For four data sets, BRN errors for other than smallest k are lower than the errors of the other methods. For Crime data, COIR gives better results.



(a) COIR (b) COIR (c) COIR

(d) mKDR (e) mKDR (f) mKDR

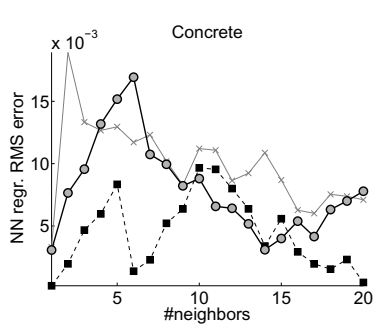
(g) BRN (h) BRN (i) BRN

(j) KDR (k) KDR (l) KDR

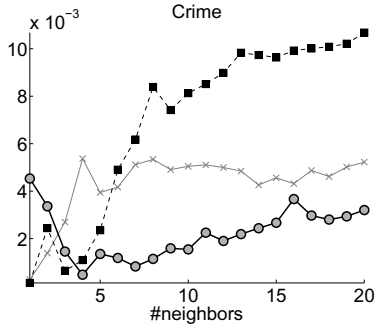
(m) SIR (n) SIR (o) SIR

(p) lin. BRN (q) lin. BRN (r) lin. BRN

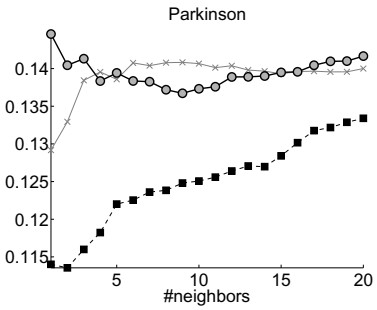
Figure 5.5. Visualizations of Head Pose (left), Yale (middle) and Crime (right) data sets. Point locations are determined by the DRR method, the shade shows the true target values. Note that in cases with two targets the point clouds are the same and only the target coloring differs, as all the methods use multivariate inputs and targets for determining a single set of low-dimensional coordinates.



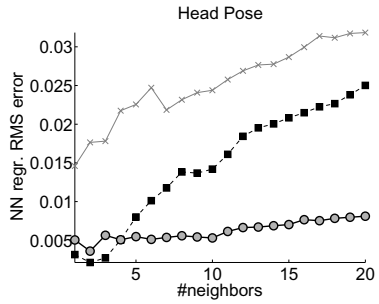
(a) Concrete



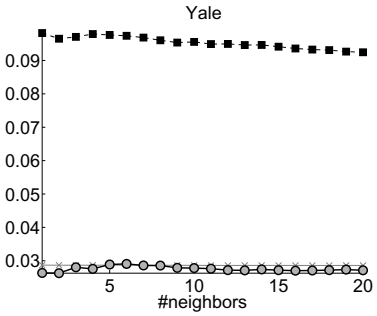
(b) Crime



(c) Parkinson

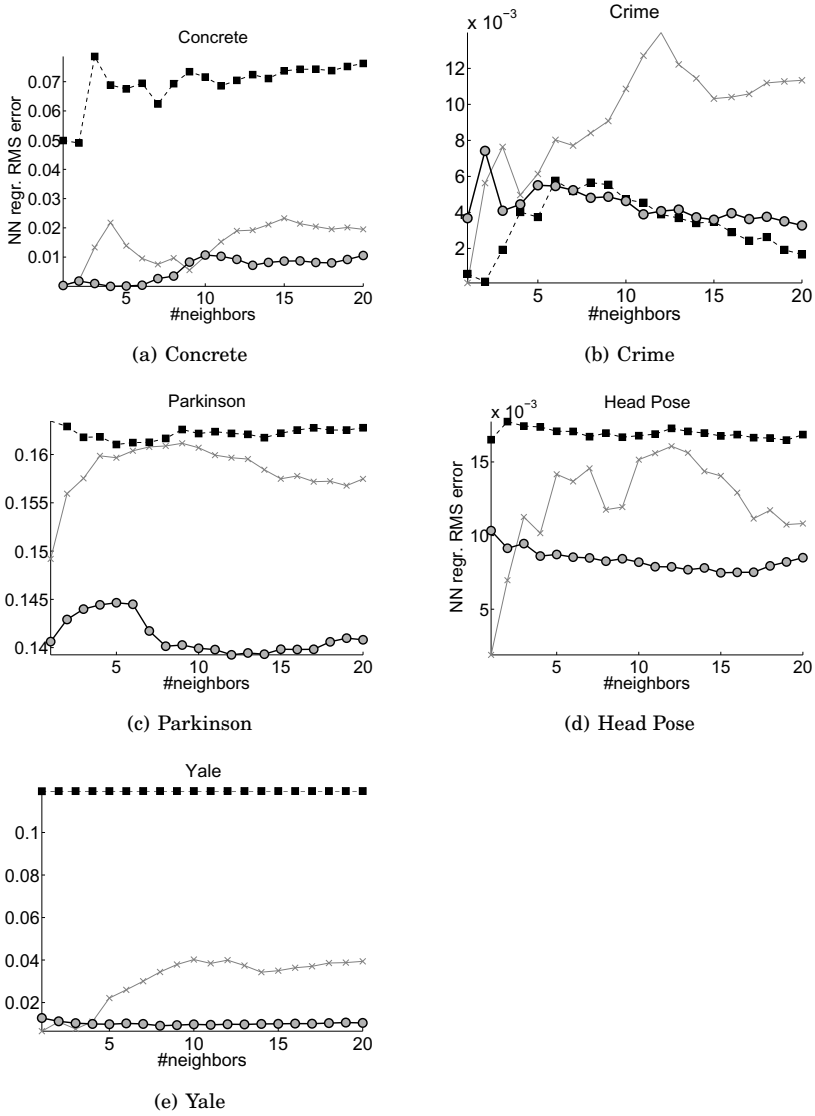


(d) Head Pose



(e) Yale

Figure 5.6. NN-regression RMS errors for the linear DRR methods.



(a) Concrete

(b) Crime

(c) Parkinson

(d) Head Pose

(e) Yale

Figure 5.7. NN-regression RMS errors for the nonlinear DRR methods.

5.5 Conclusions

In this chapter we experimented with supervised bottleneck networks for classification and regression. Our work on bottleneck classifiers serves to bring class-supervised bottleneck networks up to date. Earlier work used shallow networks and small data sets, and only compared supervised networks to unsupervised ones [84]. The only supervised dimension reduction method available for comparison at the time was LDA, a linear method. It was compared to nonlinear discriminant analysis (in terms of this work, a bottleneck classifier with linear classifier part) in [118], but the finding that a nonlinear network outperforms a linear method in classification accuracy is hardly newsworthy today. We performed a comparison to nonlinear versions of two recently developed dimension reduction methods, using deep networks.

We found bottleneck classifier to be able to compete with the two comparison methods. Based on these experiments it seems that NCA may outperform BC with larger bottleneck dimensionality, whereas BC gives better results when the bottleneck has only a few units. This suggests NCA could be a better choice for data packing, provided the quadratic training time is not a problem, and BC would be better for visualizations or other tasks where very low-dimensional representations are needed. With large data sets, the O(N) scaling of the evaluation of the cost function is a benefit over the competitors, which scale O(N^2) or worse. In addition to the number of data points, the overall training time depends on the structure of the network and the number of iterations needed to train it. In our experiments, all networks had identical encoder parts. BC has also a decoder part, but the number of weights in it is much smaller than in the encoder, so this should not introduce large differences in running times. The number of finetuning iterations was determined by early stopping, and was different in different runs. As we did not notice systematic differences between methods, we have no reason to believe that change of cost function would significantly change the number of iterations needed.

To the author's knowledge, bottleneck networks have not been applied in the field of dimension reduction for regression before. Bottleneck regression networks provide a versatile DRR method which can be used for finding both linear and nonlinear subspaces, and with linear and nonlinear response functions. Mappings that a neural network can learn can be controlled by adjusting the number and type of hidden units. Training time scales linearly in the number of data points, so BRN has potential to be used with large data sets. Neural networks are a mature and well-established class of methods, and several approaches for weight initialization, training, and regularization of weights are known.

In performance, BRN is comparable to other state-of-the-art DRR methods. For other than smallest k, BRN generally gives low k-NN-regression errors compared with the other methods. This tells about ability to preserve overall structure of data well, although at the expense of sometimes yielding higher errors if only immediate neighbors are considered.

Greatest challenge for widespread BRN use is the existence of local minima of the cost function. Energy landscapes of neural networks, especially large ones, are complex, and the cost function has several minima. In the BRN experiments we approached this problem by stochastic training and multistart strategy. While this is adequate for small networks, more complicated data sets might require larger networks, and the number of weights to be optimized could be much larger than what we have used here. This would mean slower evaluation of the cost function and neces-

sity to sample a larger energy landscape to find good minima. In this sense, BRN is also sensitive to dimensionality of input data, since each input coordinate generates a whole set of weights which connect it to the network. Bad scaling of training times as function of number of weights may cancel some benefits obtained from linear scaling in number of data points.

Two routes out from this situation are in sight: developing methods for finding good initial values for weights, which improves results of both gradient descent and stochastic methods, or using deep networks, for which successful initialization strategies [81, 102] are already known. Some strategies might work for shallow networks as well, with adequate training data available. In this work we used small data sets, that can be handled by quadratically or cubicly scaling DRR methods as well as BRN, so we did not experiment with data-intensive initialization. Problems caused by large input dimensionalities can be alleviated by preprocessing the data e.g. by principal component analysis before using it with BRN.

6. Dimension reduction with a sparse metric

The quadratic complexity of DR methods that use full pairwise (dis)similarity matrices is obviously a hindrance for using DR with large scale data. It is therefore appealing to reduce computational times by resorting to sparse methods.

We present a novel approach for doing dimension reduction using sparse matrices, which has been published in [133]. The modern trend of concentrating on local relationships easily conjures up the idea that local information should be kept, and that a global view would somehow grow out from locally correct parts. Unfortunately, this is not the case. Discarding global information, i.e. the longest distances, can ruin the embedding quality [70]. This calls for a controlled tradeoff of local and global in sparse DR methods. In Sec. 6.2 we show that a sparse version of t-SNE can be created by suitably balancing local and global information.

6.1 Existing sparse approaches

6.1.1 Dealing with naturally sparse data

Sparse data, which is regularly met in graph theory and e.g. in social network applications, is seldom used as such in DR methods. More often, even if the input data is sparse, a dimension reduction method is applied to a full distance matrix, that is created e.g. by computing shortest paths or filling in missing distances by some large value [28]. This allows using dimension reduction methods on network data, but loses the potential for faster computation, which would result from exploiting the sparse structure of data. Sparse data is used in some methods that build local views based on nearest neighbors, since such information is readily available in network data [162, 163].

6.1.2 Landmarks

Landmark algorithms are a sparse dimension reduction approach, that is able to give good global view on data, with the price of possibly sacrificing local details. The landmark idea has been used in several works in different forms. Common feature of all landmark algorithms is to designate a subset of data points as landmarks, which are treated differently from the rest of the points. Usually, the landmark points are embedded accurately, and locations for other points are decided based on the landmarks. With N data points and L landmarks this leads to complexity $O(L^2 + NL)$ in methods that use pairwise comparisons.

A sparse version of Sammon mapping [27] first embeds a subset of landmark points, and fixes their locations. Other points are placed by minimizing the Sammon cost between all (landmark, non-landmark) pairs, without considering the pairwise non-landmark distances.

More common approach for placing non-landmarks is triangulation, where the data-space distances to nearest landmarks determine weights for the landmarks, and point locations are weighted combinations of landmark locations. In Landmark MDS [37] this results in projection of non-landmark points to highest principal axes of landmarks. Another fast MDS version [50] uses a similar idea.

Point locations in landmark MVU [200] are based on a linear mapping, which as accurately as possible reconstructs the data by using L landmarks. The mapping is found from LLE-like representation of points as linear combinations of their neighbors, and the Laplacian of the resulting graph.

A different approach, slightly confusingly termed a "landmark" approach, is presented in [182]. A set of landmark points is used, but unlike in other landmark methods, only these points are embedded in the final result. The role of the other points is to provide information about data distribution, so that the embedding of the landmark points will give a representative view of all data. Information about all points is used when determining landmark locations. This happens via a random walk in a graph that has weighted connections between k-neighbors. Random walks are started from all landmark points, and ended as soon as they hit another landmark. Relative frequency of ending in landmark j when starting from i determines the neighborhood probability $p_{j|i}$ of the two landmarks. This probability is used to compute the embedding.

6.1.3 Sparse optimization updates on full matrices

It is also possible to keep the full matrix but consider only randomly chosen entries in each iteration. This speeds up computations, although it does not reduce the need for memory. A full pairwise matrix is highly redundant. When location of one point changes, N entries of the matrix are affected. This is exploited, although not in any systematic fashion, in [2] and [27]. The former chooses a random point pair in each iteration and updates their locations. Also the latter optimizes the locations of one pair at a time, but the optimization is completed and the locations fixed before a new pair is considered. In curvilinear component analysis [76] the gradient is computed and updates made w.r.t. one of the points at a time, which on average produces results like that of normal gradient descent. Extra benefit of this approach is that although the cost on average decreases, it can also temporarily increase, which should help in escaping from local minima [76].

6.1.4 Using only local distances

Accurately preserving local information is emphasized by many dimension reduction methods [148, 8, 201]. Moving into local scale happens by just considering nearest neighbors for each data point. Locally linear embedding, for one, contains such localization step, and eigendecomposition of sparse kernel matrices thus produced is much faster than decomposing full matrices.

As local approach has worked well with manifold methods, it may seem like a natural idea to keep either shortest distances or distances to k-

neighbors when creating sparse matrices for other DR methods. This idea has not proved successful. The locality-oriented manifold methods contain also a global step, which makes sure the local views are turned into a global view in a sensible way. In methods which were not originally designed to use sparse, local-only information, there is no mechanism which would guarantee global quality given only local information.

Limitations of local methods have been noticed in different works when using multidimensional scaling. The issue is studied experimentally in [70], where deleting largest entries from a pairwise distance matrix deteriorates the quality more than discarding the short distances. A localized variant of MDS produces spherical shapes unrelated to underlying structure of data, a phenomenon which is theoretically explained in [21].

6.1.5 Using randomly chosen distances

The following methods are related to our work in that some randomness is used when choosing which entries of a distance matrix to use. A sparse data version of Sammon mapping is developed in [119]. The algorithm starts with a full distance matrix, but number of nearest neighbors considered decreases in each iteration. A sparse multidimensional scaling approach with an iteratively updated set of near neighbors and randomly chosen sets of other neighbors is developed in [26, 125], and some corrections to complexity analysis are presented in [156].

6.2 Local and global in sparse distance matrices

Many dimension reduction methods use a pairwise distance matrix, and this significantly restricts the number of data points which can be embedded. It would be tempting to reduce computational burden by using only part of the distances.

Since importance of local distances is widely recognized in dimension reduction research [148, 8, 201], a natural approach to sparsifying is to keep the local, short distances and discard the long distances. However, purely local approaches to multidimensional scaling are inadequate, since removing the large distances can decrease quality more than removing the short ones [70]. This finding may have resulted in generally pessimistic views regarding sparsification of distance matrices, judging from the small number of works on this topic. Some work on partially filled matrices or naturally sparse data has been done [26, 119, 28], but most sparse dimension reduction methods are based on landmarks [37, 200], a different approach, where only a subset of points is embedded, and locations for other points are inferred from the landmark locations.

We show that part of the distances can be discarded without too much impact on quality. The sparse pairwise matrix should contain both short-scale and long-scale distances. Results from such a matrix can give much better embedding results than a purely local approach.

We parameterize the mixture of local and global distances borrowing ideas from random graph theory [198]. We see the sparse distance matrix as a graph, with a link between points if the corresponding distance is present in the matrix. Some links between near neighbors are always kept, and some links point to randomly chosen, possibly far-away, neighbors. Randomness of the graph determines, how big proportion of the links is chosen randomly. We study connection between randomness level and embedding quality, measured by trustworthiness and continuity.

6.2.1 Sparse t-SNE

We use t-SNE [182] in the experiments, but modify it to use a sparse similarity matrix.

In the following, we identify data points with graph nodes, and pairwise distances with links between nodes. N denotes the number of data points in each experiment (usually a subsample of the full data set), and L is the average number of links from a point to other points. Relationship of L and N determines *sparseness* of a matrix. We determine the sparsity structure with similar "link to nearest neighbors, then rewire" -construction as Watts and Strogatz used in their celebrated study [198] on small world graphs. The fraction of rewired links, or *randomness*, is denoted with R.

Let E denote a set of ordered pairs that contains a pair (r, c) if the similarity matrix has a non-zero element at row r, column c. Then the sparse t-SNE cost (cf. the full cost function Eq. (2.16)) becomes

$$C_{\text{sparse}} = \sum_{(i,j)\in E} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{6.1}$$

Since E has (roughly, see below) NL elements, this changes the time complexity from $O(N^2)$ to $O(NL)$.

The set E is built as follows. First, approximately L nearest neighbors c_j are listed for each point r. The number of neighbors is approximate, because the neighborhood relationships are not symmetric, and we need a symmetric matrix. We use $a < L$ neighbors, but insert both (r, c_j) and (c_j, r) into E. The value a is chosen by a rough rule of thumb, which ignores the distribution of data and basicly assumes that having a connection from A to B does not affect the probability of B being connected to A. If we use a neighbors out of N possible, then A will connect to B with probability $\frac{a}{N}$, B will connect to A equally often, and some of the links will be two-directional. We want to link the two points with probability $\frac{L}{N}$, so the necessary value of a can be solved from $\frac{2a}{N} - (\frac{a}{N})^2 = \frac{L}{N}$, giving $a = N - \sqrt{N^2 - NL}$.

After inserting the nearest neighbor links into E, we change part of them into long distance links. The set E has $M \approx NL$ entries. We delete RM elements from the set, with uniform probability. This means undoing some work from the previous step, but this two-step procedure is conceptually simple, easy to implement and allows reusing the neighbor lists for different values of R. We pick r and c from range $1, \ldots, N$, insert pairs (r, c) and (c, r) into E if $r \neq c$ and $(r, c) \notin E$, and repeat this until we have created RM new links. Most links thus created will be long-distance links, simply because there usually are much more long-distance than short-distance neighbors available.

The original t-SNE determines kernel widths from a perplexity parameter, which is essentially a soft number of nearest neighbors. This means that widths can differ for different points. Determining kernel widths when only sparse matrices are available is not straightforward, and results might be affected by varying widths. Because of this we use ϵ-neighborhoods, using same fixed kernel width everywhere. This lowers overall quality, but is suitable in an experimental setting, since we focus on randomness effects and not on absolute quality. Developing sparse t-SNE into a usable method, complete with a way of choosing kernel widths in the sparse setting, is left for future work.

Another experimental feature is the use of a full distance matrix for finding the nearest neighbors. This is done for making sure we do not

introduce unexpected side effects by any approximate nearest-neighbor schemes. In real use, a full matrix would not be used as an intermediate step. Instead, the nearest neighbors would be found with an approximate method, or the probability of linking two points would be based on their distance.

6.2.2 Experiments

We use three data sets in the experiments. We will only show results for MNIST data set but qualitatively similar results (with somewhat smaller sample sizes and smaller parameter ranges) were obtained using two more data sets, the USPS and Yale data. See Appendix A for details of the data sets.

We create a continuum of sparse matrices by varying L and R, and study the effects of randomness and sparsity on visualization results. We measure visualization quality by trustworthiness and continuity. Trustworthiness and continuity values at 5-neighborhoods are used.

6.2.3 Effect of randomness

In the first experiment, we look at trustworthiness and continuity as function of randomness. We repeat the experiment for several sparseness levels, by fixing L and letting N change.

The regular connection pattern $R = 0$ % gives poor results. Trustworthiness levels around 0.5 are typical when a random set of points is compared to the original data. This is seen both in the 2D visualization in Fig. 6.1 an in the numerical results in Fig. 6.2.

When R increases, results get more trustworthy. Fig. 6.2 shows an upward sloping line, and Fig. 6.1 has more clearly separated classes. When matrices are not very sparse (cases N=2000 and N=4000), the completely random matrix gives best results. When matrices are sparser, best results are obtained with $R = 80$ %. Continuity, on the other hand, stays at roughly the same level or only slightly increases, up to $R = 80$ %. When the matrix is made completely random, continuity drops. Matrix sparseness affects the steepness of the drop.

These observations are in line with the nature of the two criteria. Achieving high trustworthiness requires global information, i.e. long distance links, because different neighborhoods must be kept separate in the visualizations. Before rewiring, the nearest neighbor matrix has only local links. Each random link is a potential long distance link; therefore increasing proportion of random links increases trustworthiness. Continuity, more easily created by local constraints, is much less sensitive to matrix randomness. As long as points from a neighborhood stay together the result is continuous; it does not matter if the points mix with other neighborhoods.

High quality with $R = 80$ % and a drop with $R = 100$ % brings up an important point regarding the connection patterns: although global links are important, also enough local information must be provided. Missing local information is suggested by different behavior of the continuity and trustworthiness criteria. Continuity, associated with local constraints, drops very clearly, whereas the more global trustworthiness is not affected as drastically.

Existence or steepness of the drop depends on matrix sparseness. When the matrix is not very sparse, some random links will necessarily be between near neighbors. This provides some local information, even in com-

pletely random matrices. The sparser the graph, the smaller the probability of hitting a pair of near neighbors by chance. Therefore, effects of having an insufficient number of local links are seen more clearly in sparser matrices.

6.2.4 Effect of sparseness

Our second experiment fixes randomness and lets the sparseness change. We use randomness $R = 80\ \%$, which gave the best results in the first experiment.

Our most important discovery is the form of dependence of trustworthiness on sparseness in Fig. 6.4. Trustworthiness increases rapidly in the beginning, but growth slows down when L approaches N. For $N = 3000$, using 2000 neighbors gives the same trustworthiness as the full matrix. Although numerical results for $L = 1000$ are lower, in the illustration in Fig. 6.3 it is difficult to visually tell the $L = 3000$ and $L = 1000$ results apart. Lines for larger N in Fig. 6.4 have not reached a constant level within the L range used, but also their growth decelerates with growing L. This suggests that the number of neighbors needed for certain quality can be a sublinear function of N.

This has promising practical implications. A sparse dimension reduction method is really useful only, if a reduction in computational complexity is achieved. A method scaling $O(NL)$ is still quadratic, if we must use an L which a linear function of N. Figs. 6.3 and 6.4 suggest that for satisfactory results with sparse t-SNE, L may grow more slowly than N. This makes the overall complexity subquadratic.

6.3 Conclusions

The idea of reducing computation times of dimension reduction methods by discarding part of pairwise distances has been considered in the literature. The natural approach is to favor local accuracy by using only distances to nearest neighbors. It has been shown, however, that large distances have a large impact on quality, and therefore cannot be discarded. This has resulted in preconception that creating sparse dimension reduction methods by keeping all points but only part of distances would be next to futile.

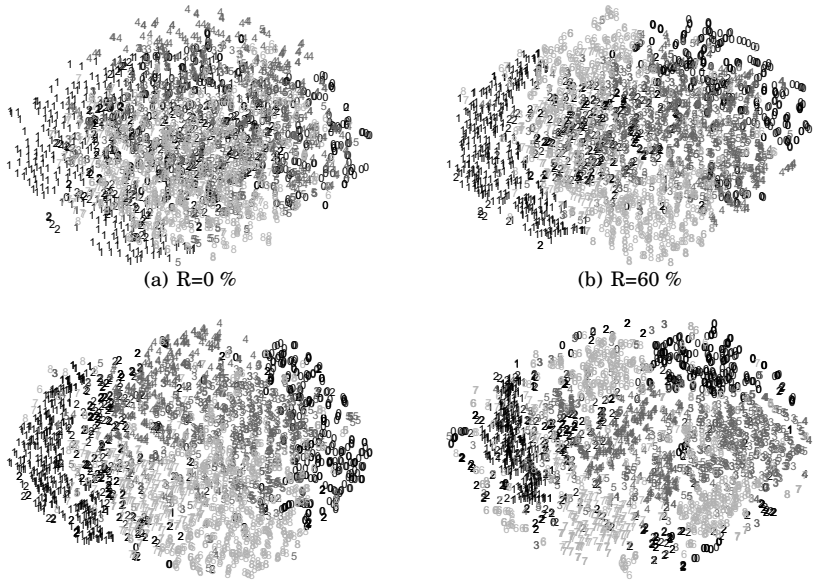
In this work we brought new light on this overly simplified view. We pointed out that keeping the nearest neighbors is not the only way to sparsify a distance matrix. Indeed, it is much more advisable to replace some local links with random links. Proportion of random links has a clear impact on results.

We discussed the effect of randomness, changing from a matrix with purely local links to a matrix with purely randomly chosen neighbors. Effects on trustworthiness and continuity of visualizations were measured. We found a general connection between increasing randomness and increasing trustworthiness. Completely random matrices do not work well, however. Trustworthiness is lower for completely random than highly random matrices. Continuity stays high until high randomness levels, but drops sharply for completely random matrices. This effect is reduced if the matrix is not very sparse. In such case, also random links will sometimes be between near neighbors, which helps to maintain high continuity.

Sparseness seems to have little effect on continuity of visualizations.

Trustworthiness is affected, but we made a promising observation that the number of links needed for certain quality seems to be a sublinear function of N. This makes the overall operation subquadratic, an improvement over the quadratic full-matrix version of t-SNE. The lower complexity with not too much decrease in quality could stretch the feasible range of t-SNE use from thousands of points to at least tens of thousands.

Some work remains to be done before sparse t-SNE is ready for serious use. For this work we implemented an experimental version, which used simplified parameterization (same kernel width everywhere). The nearest neighbor approach we used for creating the sparse matrices forms a slow preprocessing step, and forces a somewhat artificial division into local and global links. While the experimental version was adequate to studying randomness effects, a sampling-based approach, that chooses links by certain probability depending on their length, will probably be better for real use.



(a) R=0 %

(b) R=60 %

(c) R=80 %

(d) R=100 %

Figure 6.1. 3000 MNIST points, with $L = 450$ and varying R. Colors denote digit classes.

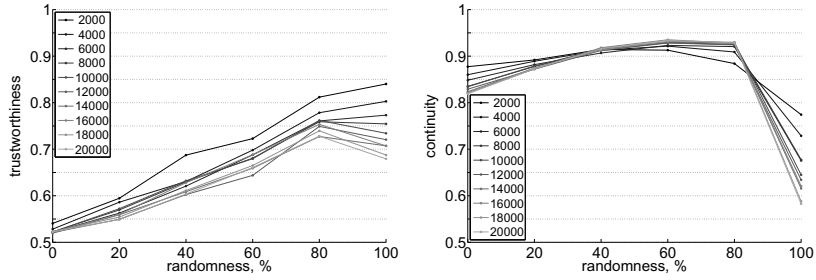
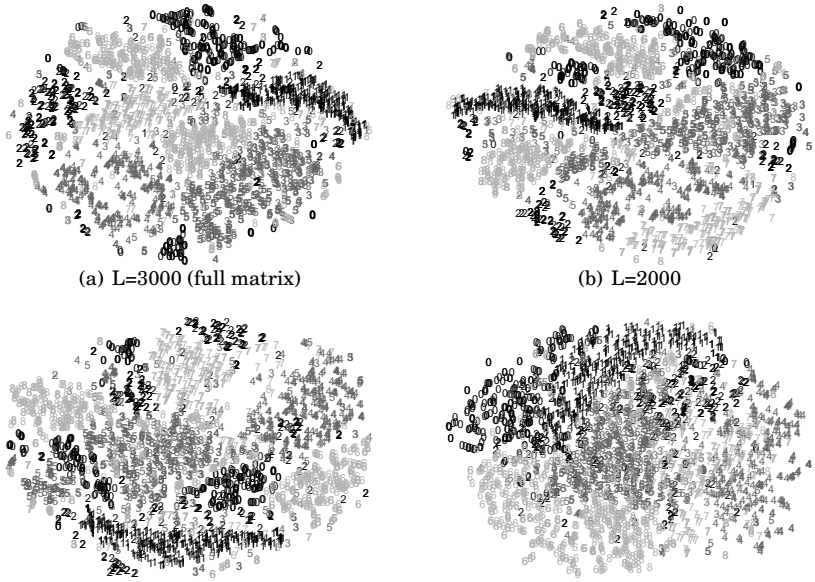


Figure 6.2. Effect of randomness level, for different N and L=200.



(a) L=3000 (full matrix) (b) L=2000

(c) L=1000 (d) L=300

Figure 6.3. 3000 points of MNIST data, with different average neighbor numbers. Randomness level is 80 %.

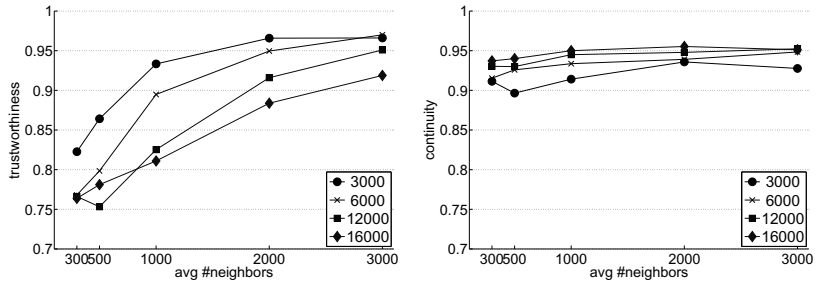


Figure 6.4. Effect of average number of neighbors, for different N, with $R = 80$ %.

7. Discussion

Although predominantly a dimension reduction study, this work contributed also to two current topics of neural network research: deep networks and networks that use random projections. We also promoted a neural network approach in dimension reduction for regression. DRR is a subfield of dimension reduction research, where neural methods are to our knowledge currently not used, although similar architectures have been applied in other fields.

Deep bottleneck classifiers we studied in Chapter 5 complement current research on deep autoencoders and deep supervised networks that use quadratically scaling cost functions. Bottleneck networks are an attractive tool for dimension reduction, since evaluation of their cost function scales linearly in the number of data points, and since they naturally specify a mapping from data space to low-dimensional space. The cost functions of many supervised DR methods scale quadratically or worse, and many cannot determine locations for test points, or do so only approximately.

In Chapter 4 we studied extreme learning machines (ELM). ELM is a relatively new method, and initial hype, easily caused by novel ideas, is not over yet. There is already a lot of research on ELM, but much of it is incremental, combinatorial, or applied. ELMs are being trained with evolutionary algorithms, collected into ensembles, used in fuzzy fashion, refined by pruning hidden units – soon ELM will have been combined with every single machine learning technique ever invented. More analytical ELM works are scarce, however. We presented a novel interpretation of ELM, pointing out its resemblance to kernel classifiers. We also discussed some claims made about ELM, which we feel may have been too strongly presented in earlier work. A point with practical consequencies to ELM training is the role of variance of weights in the randomly weighted hidden layer. We found the variance to affect classification results. We therefore emphasize that weight variance should be a tuning parameter of ELM, and not an arbitrarily fixed constant as is the current practice.

Chapters about bottleneck networks and extreme learning machines presented studies, which are in some sense closed: they studied a problem, answered some questions, and were finished. While there is always room for improvement and new views, no obvious follow-up problems were left dangling. Chapters about model-based metrics and sparse dimension reduction, on the contrary, open some potential directions for further study.

Cophenetic distances with Sammon mapping proved a viable tool for visualizing hierarchical structures in Chapter 3. We presented an exam-

ple of visualizing hierarchical clustering results on a small data set. The point in this work was to use the visualization for presenting results in a clustering problem, as an alternative to e.g. dendrograms or other tree-shaped visualizations. Hierarchical approach, however, may be useful also in problems were the main goal is not clustering, but simply presentation of data. Data sets that are studied today can contain millions of points, which is too much information to be shown at once. Arranging data hierarchically enables controlling the level of detail, and zooming into areas of interest. Hierarchical clustering is already possible on large data sets. If large-data versions of Sammon mapping could be created, for example by using sparse matrix technique from Chapter 6, cophenetic Sammon mapping could become a new tool for visualizing large data sets.

Successful use of Sammon mapping with a clustering metric draws new attention to distance-preserving DR methods. The modern trend is to emphasize local neighborhoods, since showing them accurately is thought to give reliable view on data. But, if the view on the data has been built into a metric, like in the case of cophenetic distances, then showing the metric as accurately as possible becomes important. In such cases, distance-preserving methods may outperform neighborhood-oriented methods. Although many dimension reduction methods are combinations of a metric and a cost function, the interplay of the two warrants further study.

Another metric we used with dimension reduction was derived from the feature space of a neural network model. In this work we mainly used the model-based metric for creating supervised variants of unsupervised DR methods (Chapter 3). As demonstrated by visualizations of feature spaces in Chapter 4, model-based metrics can also be used for studying the model. One area where visualizations of feature spaces of neural networks might prove useful are deep networks. Understanding the effects of pretraining and the nature of mappings that the layers learn is currently an important research topic. Layerwise visualizations of feature spaces might shed new light to these questions.

The work on sparse dimension reduction in Chapter 6 is just started. We showed that, contrary to the current belief, using sparse matrices for creating fast dimension reduction methods is a feasible idea. This brings up the intriguing possibility to apply sparse t-SNE in naturally sparse data, abounding e.g. in social networks research. We already learned that a proper balance between short-distance and long-distance links is needed for good results, but we do not yet know how the proportion should be chosen, and which other factors have an effect. From technical point of view, the sparse version of t-SNE we used in this work is incomplete and rather clumsy to use, and needs reworking. Developing sparse dimension reduction further along the lines sketched in this work is a promising direction for further research.

A. Data and parameters

A.1 Description of data sets

Table A.1. Binary classification data sets (from UCI repository)

name	# samples	# dims	data types
Arcene [71]	200	10000	cont.
US votes	435	16	bin.
WDBC	569	30	cont.
Pima	768	8	cont.
Tic Tac Toe	958	27	categ.
Internet ads	2359	1558	cont., bin.

Table A.2. Multiclass classification (available from `http://www.cs.nyu.edu/~roweis/data.html`)

data set	dimensions	classes
MNIST	28x28	10
USPS	16x16	10
20 newsgroups	100	4

The MNIST data contains 70000 digit images of 28x28 pixels, with roughly equal numbers of digits from each class. USPS contains handwritten digits from US Postal Service. USPS data has 1100 16x16 pixel images from each class. The newsgroups data consists of 16242 documents collected from internet newsgroups, with binary occurrence vectors for 100 chosen words. The documents are classified into four broad categories (comp.*, rec.*, sci.*, talk.*).

"UCI" as the source refers to the UCI Machine Learning Repository [56] `http://archive.ics.uci.edu/ml/` and "ExtYaleB", the Extended Yale Face Database B, is available from `http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html`. The table lists the dimensions as used in the experiments; in some cases the original data was preprocessed (smaller images, dimensions with missing data dropped etc).

Two of the data sets are face images, one (Head Pose data) where the task is predicting the horizontal and vertical pose of the head, and one (Yale data) where the target variable is direction (azimuth and elevation) of illumination. Three other data sets are smaller dimensional. The

Table A.3. Regression data sets

name	targets	data dim	#train	#test	data source	ref.
Head Pose	2	64x64	349	349	isomap.stanford.edu/datasets.html	
Yale small	2	48x64	2419	1728	ExtYaleB	[61]
Parkinson	2	16	4165	1710	UCI	[176]
Crime	1	99	1396	598	UCI	[147]
Concrete	1	8	500	50	UCI	[211]

covariates are different measurements or statistics, and the prediction tasks are two disease symptom scores (Parkinson data), number of violent crimes per population (Crime data) and compressive strength (Concrete data).

A.2 Data and parameters of experiments

Visualizing hierarchical clustering results (Sec. 3.2)

USPS, with 100 samples from each class. For more detail, see [132].

Studying ELM behavior (Chapter 4)

Arcene, US votes, WDBC, Pima, TicTacToe, Internet ads. Data are scaled to range $[-1, 1]$. Each data set is divided into 10 parts. Nine parts are used for training and one for testing, repeating this 10 times. Five different deviations ($\sigma \in \{0.1, 0.325, 0.55, 0.775, 1\}$) are used. For more detail, see [134].

Visualizing MLP features (Sec. 3.1.2)

Arcene, US votes, WDBC, Pima, TicTacToe, Internet ads.

Sparse t-SNE (Sec. 6.2

MNIST, USPS, Yale. We sampled random subsets from these data to create matrices of varying sizes. For more detail, see [133].

Bottleneck regression networks (Sec. 5.4)

Head Pose, Yale, Parkinson, Crime, Concrete. Target variables are scaled to [0,1]. The input data are centered at 0.

Initial weights for BRN are drawn from a zero-mean normal distribution with deviation 0.01. The cooling schedule of simulated annealing is to set new temperature to 80 % of the old, and proposals for new weight vectors are obtained by adding small normally distributed random numbers to all weight vector entries.

Numbers of hidden units are fixed to 50 for L1 and 10 for L3. We use fixed values based on the finding that BRN is not very sensitive to number of hidden units.

For more detail, see [131].

Bottleneck classifiers (Sec. 5.3)

MNIST, USPS, 20 newsgroups.

The MNIST data set has a fixed division into training data (60000 samples) and test data (10000 samples), which was respected also in our experiments. USPS was divided into training data of 9000 samples and test data of 2000 samples. In 20 newsgroups data, we used 12000 randomly chosen documents for training and 4200 for testing (for easier division into training batches some documents were left out).

We used 1000-point batches for finetuning MNIST and USPS, and 800-point batches for newsgroups. Part of training samples (10 %) was used as validation data for choosing parameter values and for determining the number of finetuning epochs with early stopping.

For more detail, see [130].

Table A.4. Parameter values used in method comparisons. BC L5 size gives the number of hidden units in BC classifier layer.

	MNIST	USPS	newsgroups
encoder layout	$1000 \times 500 \times 250$	$400 \times 200 \times 100$	$100 \times 75 \times 50$
BC L5 size	2D: 210, 30D: 130	2D: 10, 30D: 90	2D: 10, 30D: 170
t-SNE perplexity	2D: 10, 30D: 50	2D: 10, 30D: 130	2D: 110, 30D: 10
LMNN neighbors	$k = 3$, $m = 10$	$k = 3$, $m = 10$	$k = 3$, $m = 10$
pretrain epochs	50	50	50

Bibliography

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc. of ICDT*, volume 1973 of *LNCS*, pages 420–434, 2001.

[2] D. K. Agrafiotis. Stochastic proximity embedding. *Journal of Computational Chemistry*, 24(10):1215–1221, 2003.

[3] B. Alewijnse, J. Nerbonne, L. J. van der Veen, and F. Manni. A computational analysis of Gabon varieties. In *Proc. of RANLP workshop on computational phonology*, pages 3–12, 2007.

[4] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

[5] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[6] G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Computation*, 12:2385–2404, 2000.

[7] H.-U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, 1992.

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

[9] M. Belkin and P. Niyogi. Towards a theoretical foundation for Laplacian-based manifold methods. *Journal of Computer and System Sciences*, 74:1289–1308, 2008.

[10] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–27, 2009.

[11] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16:2197–2219, 2004.

[12] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Proc. of NIPS*, volume 19, pages 153–160. MIT Press, 2007.

[13] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-scale kernel machines*, Neural Information Processing Series, chapter 14, pages 321–359. MIT Press, 2007.

[14] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and spectral clustering. In *Proc. of NIPS*, volume 16, 2004.

[15] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of ICDT*, volume 1540 of *LNCS*, pages 217–235, 1998.

[16] J. C. Bezdek and N. R. Pal. An index of topological preservation for feature extraction. *Pattern Recognition*, 28(3):381–391, 1995.

[17] C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

[18] R. Boardman. Bubble trees: the visualization of hierarchical information structures. In T. Turner and G. Szwillus, editors, *Extended abstracts of ACM conference on Human Factors in Computing Systems*, pages 315–316. ACM Press, USA, 2000.

[19] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling*. Springer Series in Statistics. Springer, 2nd edition, 2005.

[20] M. Brand. Charting a manifold. In *Proc. of NIPS*, volume 15, 2002.

[21] A. Buja, B. F. Logan, J. A. Reeds, and L. A. Shepp. Inequalities and positive-definite functions arising from a problem in multidimensional scaling. *The Annals of Statistics*, 22(1):406–438, 1994.

[22] K. Bunte, B. Hammer, T. Villmann, M. Biehl, and A. Wismüller. Neighbor embedding XOM for dimension reduction and visualization. *Neurocomputing*, 74(9):1340–1350, 2011.

[23] C. J. C. Burges. Geometric methods for feature extraction and dimensional reduction. In L. Rokach and O. Maimon, editors, *Data mining and knowledge discovery handbook: A complete guide for practitioners and researchers*, chapter 1. Kluwer Academic Publishers, 2005.

[24] M. Á. Carreira-Perpiñán. A review of dimension reduction techniques. Technical Report CS-96-09, University of Sheffield, UK, 1996.

[25] M. Á. Carreira-Perpiñán. The elastic embedding algorithm for dimensionality reduction. In *Proc. of ICML*, 2010.

[26] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proc. of IEEE Visualization '96*, pages 127–132, 1996.

[27] C. L. Chang and R. C. T. Lee. A heuristic relaxation method for nonlinear mapping in cluster analysis. *IEEE Transactions on Systems, Man and Cybernetics (part B)*, 3(2):197–200, 1973.

[28] L. Chen and A. Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104:209–219, 2009.

[29] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. Similarity-based classification: Concepts and algorithms. *Journal of Machine Learning Research*, 10:747–776, 2009.

[30] Y. Cho and L. K. Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Proc. of NIPS*, volume 22, pages 342–350, 2009.

[31] F. R. K. Chung. *Spectral graph theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[32] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *PNAS*, 102(21):7426–7431, 2005.

[33] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[34] T. M. Cover. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(1):50–55, 1968.

[35] T. Cribbin. Visualising the structure of document search results: A comparison of graph theoretic approaches. *Information Visualization*, 9(2):83–97, 2010.

[36] D. de Ridder and R. P. W. Duin. Sammon's mapping using neural networks: A comparison. *Pattern Recognition Letters*, 18:1307–1316, 1997.

[37] V. de Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, 2004.

[38] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Proc. of NIPS*, 2003.

[39] P. Demartines and J. Hérault. Vector quantization and projection neural network. In J. Mira, J. Cabestany, and A. G. Prieto, editors, *Proc. of IWANN*, volume 686, pages 328–333, 1993.

[40] 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, 1997.

[41] D. DeMers and G. Cottrell. Non-linear dimensionality reduction. In *Proc. of NIPS*, pages 580–587, 1993.

[42] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[43] C. Ding, X. He, and H. D. Simon. On the quivalence of nonnegative matrix factorization and spectral clustering. In *Proc. of SDM*, pages 606–610, 2005.

[44] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *PNAS*, 100(10):5591–5596, 2003.

[45] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2001.

[46] J. L. Elman and D. Zipser. Learning the hidden structure of speech. *Journal of the Acoustic Society of America*, 83(4):1615–1626, 1988.

[47] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

[48] P. A. Estévez and A. M. Chong. Geodesic nonlinear mapping using the neural gas network. In *Proc. of IJCNN*, pages 3287–3294, 2006.

[49] P. A. Estévez and C. J. Figueroa. Online data visualization using the neural gas network. *Neural Networks*, 19:923–934, 2006.

[50] C. Faloutsos and K.-I. Lin. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. of ACM SIGMOD international conference on Management of data*, pages 163–174, 1995.

[51] A. Fischer and C. Igel. Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In *Proc. of ICANN*, volume 6354 of *LNCS*, pages 208–217. Springer, Heidelberg, 2010.

[52] R. A. Fisher. The use of multiple measures in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[53] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[54] I. K. Fodor. A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, USA, 2002.

[55] D. François, V. Wertz, and M. Verleysen. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering*, 19(7):873–886, 2007.

[56] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[57] B. Frénay and M. Verleysen. Using SVMs with randomised feature spaces: an extreme learning approach. In *Proc. of ESANN*, pages 315–320, 2010.

[58] K. Fukumizu, F. R. Bach, and M. I. Jordan. Kernel dimension reduction in regression. *The Annals of Statistics*, 37(4):1871–1905, 2009.

[59] P. Gallinari, S. Thiria, F. Badran, and F. Fogelman-Soulie. On the relations between discriminant analysis and multilayer perceptrons. *Neural Networks*, 4:349–360, 1991.

[60] 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)*, 35(6), 2005.

[61] A. S. Georghiades, P. N. Belhumeur, and D. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):643–660, 2001.

[62] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *Proc. of NIPS*, volume 18, pages 451–458, Cambridge, MA, USA, 2005. MIT Press.

[63] A. Globerson and S. Roweis. Visualizing pairwise similarity via semidefinite programming. In *Proc. of AISTATS*, pages 139–146, 2007.

[64] A. Globerson and N. Tishby. Sufficient dimensionality reduction. *Journal of Machine Learning Research*, 3:1307–1331, 2003.

[65] C. Godsil and G. Royle. *Algebraic graph theory*. Number 207 in Graduate texts in mathematics. Springer, 2001.

[66] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *Proc. of NIPS*, volume 17, pages 513–520, 2005.

[67] G. H. Golub and C. F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[68] G. J. Goodhill and T. J. Sejnowski. Quantifying neighbourhood preservation in topographic mappings. In *Proc. of 3nd Joint Symposium on Neural Computation*, volume 6, pages 61–82, Pasadena, CA, USA, 1996. California Institute of Technology.

[69] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3 and 4):325–338, 1966.

[70] J. Graef and I. Spence. Using distance information in the design of large multidimensional scaling experiments. *Psychological Bulletin*, 86(1):60–66, 1979.

[71] I. Guyon, S. R. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Proc. of NIPS*, 2004.

[72] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2/3):197–145, 2001.

[73] J. Ham, D. D. Lee, S. Mika, and B. Schölkopf. A kernel view of th dimensionality reduction of manifolds. In *Proc. of ICML*, 2004.

[74] X. He, D. Cai, S. Yan, and H.-J. Zhang. Neighborhood preserving embedding. In *Proc. of ICCV*, 2005.

[75] X. He and P. Niyogi. Locality preserving projections. In *Proc. of NIPS*, 2003.

[76] J. Hérault, C. Jausions-Picaud, and A. Guérin-Dugué. Curvilinear component analysis for high-dimensional data representation: I. theoretical aspects and practical use in the presence of noise. In *Proc. of IWANN*, LNCS, pages 635–644, 1999.

[77] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proc. of VLDB (International Conference on Very Large Databases)*, 2000.

[78] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Proc. of NIPS*, pages 857–864, 2003.

[79] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[80] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[81] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[82] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.

[83] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006.

[84] N. Intrator and S. Edelman. Learning low-dimensional representations via the usage of multiple-class labels. *Network: Computation in Neural Systems*, 8(3):259–281, 1997.

[85] T. Iwata, K. Saito, N. Ueda, S. Stromsten, T. L. Griffiths, and J. B. Tenenbaum. Parametric embedding for class visualization. *Neural Computation*, 19:2536–2556, 2007.

[86] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Proc. of NIPS*, volume 11, 1998.

[87] D. W. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000.

[88] E. Jakkula, K. Rehnström, T. Varilo, O. P. Pietiläinen, T. Paunio, N. L. Pedersen, U. deFaire, M.-R. Järvelin, J. Saharinen, N. Freimer, S. Ripatti, S. Purcell, A. Collins, M. J. Daly, A. Palotie, and L. Peltonen. The genome-wide patterns of variation expose significant substructure in a founder population. *The American Journal of Human Genetics*, 83:787–794, 2008.

[89] N. Japkowicz, S. J. Hanson, and M. A. Gluck. Nonlinear autoassociation is not equivalent of PCA. *Neural Computation*, 12:531–545, 2000.

[90] L. O. Jimenez and D. A. Landgrebe. Supervised classification in high-dimensional space: Geometrical, statistical and asymptotical properties of multivariate data. *IEEE Transactions on Systems, Man and Cybernetics (part C: Applications and reviews)*, 28(1):39–54, 1998.

[91] S. Kaski, J. Nikkilä, M. Oja, J. Venna, P. Törönen, and E. Castrén. Trustworthiness and metrics in visualizing similarity of gene expression. *BMC Bioinformatics*, 4(48), 2003.

[92] S. Kaski, J. Sinkkonen, and A. Klami. Discriminative clustering. *Neurocomputing*, 69:18–41, 2005.

[93] M. Kim and V. Pavlovic. Dimensionality reduction using covariance operator inverse regression. In *Proc. of CVPR*, 2008.

[94] M. Kim and V. Pavlovic. Covariance operator based dimensionality reduction with extension to semi-supervised learning. In *Proc. of AISTATS*, volume 5 of *JMLR: W&CP*, pages 280–287, 2009.

[95] T. Kohonen. The self-organizing map. *Neurocomputing*, 21:1–6, 1998.

[96] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proc. of ICML*, 2002.

[97] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[98] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.

[99] K. J. Kurtz. The divergent autoencoder (DIVA) model of category learning. *Psychonomic Bulletin & Review*, 14(4):560–576, 2007.

[100] J. Lafferty and G. Lebanon. Information diffusion kernels. In *Proc. of NIPS*, volume 15, 2002.

[101] S. Lafon and A. B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1393–1403, 2006.

[102] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[103] N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005.

[104] N. D. Lawrence and J. Quiñonero-Candela. Local distance preservation in the GP-LVM through back constraints. In *Proc. of ICML*, pages 513–520, 2006.

[105] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[106] J. A. Lee, C. Archambeau, and M. Verleysen. Locally linear embedding versus Isotop. In *Proc. of ESANN*, pages 527–534, 2003.

[107] J. A. Lee, A. Lendasse, and M. Verleysen. Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis. *Neurocomputing*, 57:49–76, 2004.

[108] J. A. Lee and M. Verleysen. Nonlinear projection with the Isotop method. In *Proc. of ICANN*, volume 2415 of *LNCS*, pages 933–938, 2002.

[109] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, 2007.

[110] J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72:1431–1443, 2009.

[111] J. A. Lee and M. Verleysen. Simbed: similarity-based embedding. In C. Alippi, M. M. Polycarpou, C. Panayiotou, and G. Ellinas, editors, *Proc. of ICANN*, volume 5769 of *LNCS*, pages 95–104. Springer, 2009.

[112] E. Levina and P. J. Bickel. Maximum likelihood estimation of intrinsic dimension. In *Proc. of NIPS*, volume 17, pages 777–784, 2005.

[113] K.-C. Li. Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):316–327, 1991.

[114] H.-T. Lin and C.-J. Lin. A study of sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Unpublished manuscript, available from `http://www.csie.ntu.edu.tw/∼cjlin/tanh.pdf`, 2003.

[115] D. Lowe and M. E. Tipping. NeuroScale: novel topographic feature extraction using RBF networks. In *Proc. of NIPS*, volume 9, pages 543–549, 1997.

[116] P. C. Mahalanobis, R. C. Bose, and S. N. Roy. Normalisation of statistical variates and the use of rectangular co-ordinates in the theory of sampling distributions. *Sankhyā*, 3:1–40, 1937.

[117] J. Mao and A. K. Jain. Discriminant analysis neural networks. In *Proc. of Int. Conf. Neural Networks*, volume 1, pages 300–305, 1993.

[118] J. Mao and A. K. Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, 6(2):296–317, 1995.

[119] M. Martín-Merino and A. Muñoz. A new Sammon algorithm for sparse data visualization. In *Proc. of ICPR*, 2004.

[120] P. McCullagh and J. A. Nelder. *Generalized linear models*, volume 37 of *Monographs on statistics and applied probability*. Chapman & Hall, 2nd edition, 1989.

[121] M. J. McGuffin and J.-M. Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, 2009.

[122] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. OP-ELM: Optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, 2010.

[123] R. Min, D. A. Stanley, Z. Yuan, A. Bonner, and Z. Zhang. A deep non-linear feature mapping for large-margin kNN classification. In *Proc. of ICDM*, 2009.

[124] R. Min, L. van der Maaten, Z. Yuan, A. Bonner, and Z. Zhang. Deep supervised t-distributed embedding. In *Proc. of ICML*, 2010.

[125] A. Morrison, G. Ross, and M. Chalmers. A hybrid layout algorithm for sub-quadratic multidimensional scaling. In *Proc. of IEEE Symposium on Information Visualization*, 2002.

[126] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis. Diffusion maps, spectral clustering and the reaction coordinates of dynamical systems. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[127] J. Nilsson, F. Sha, and M. I. Jordan. Regression on manifolds using kernel dimension reduction. In *Proc. of ICML*, 2007.

[128] A. Noack. An energy model for visual graph clustering. In *Proc. of GD*, pages 425–436, 2003.

[129] V. Onclinx, V. Wertz, and M. Verleysen. Nonlinear data projection on non-Euclidean manifolds with controlled trade-off between trustworthiness and continuity. *Neurocomputing*, 72:1444–1454, 2009.

[130] E. Parviainen. Deep bottleneck classifiers in supervised dimension reduction. In *Proc. of ICANN*, volume 6354 of *LNCS*, pages 1–10. Springer, 2010.

[131] E. Parviainen. Dimension reduction for regression with bottleneck neural networks. In C. Fyfe, P. Tino, D. Charles, C. Garcia-Osorio, and H. Yin, editors, *Proc. of IDEAL*, volume 6283 of *LNCS*, pages 37–44. Springer, 2010.

[132] E. Parviainen. Reliability of dimension reduction visualizations of hierarchical structures. In *Proc. of ESANN*, pages 105–110. d-side publications, 2010.

[133] E. Parviainen. Effects of sparseness and randomness of pairwise distance matrices on t-SNE results. In *Proc. of ESANN*, pages 357–362. d-side publications, 2011.

[134] E. Parviainen, J. Riihimäki, Y. Miche, and A. Lendasse. Interpreting Extreme Learning Machine as an approximation to an infinite neural network. In *Proc. of KDIR*, pages 65–73. INSTICC, 2010.

[135] E. Parviainen and A. Vehtari. Features and metric from a classifier improve visualizations with dimension reduction. In C. Alippi, M. M. Polycarpou, C. Panayiotou, and G. Ellinas, editors, *Proc. of ICANN*, volume 5769 of *LNCS*, pages 225–234. Springer, 2009.

[136] E. Parviainen and A. Vehtari. Explaining classification by finding response-related subgroups in data. In *Proc. of SNPD 2010 (ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing)*, pages 69–75. IEEE Computer Society, 2010.

[137] M. S. Park and J. Y. Choi. Theoretical analysis on feature extraction capability of class-augmented PCA. *Pattern Recognition*, 42:2353–2362, 2009.

[138] E. Pękalska and R. P. W. Duin. Learning with general proximity measures. In *Proc. of PRIS*, 2006.

[139] J. Peltonen, J. Goldberger, and S. Kaski. Fast discriminative component analysis for comparing examples. In *Proc. of NIPS*, 2006.

[140] J. Peltonen and S. Kaski. Discriminative components of data. *IEEE Transactions on Neural Networks*, 16:68–83, 2005.

[141] J. Peltonen, A. Klami, and S. Kaski. Learning more accurate metrics for self-organizing maps. In J. R. Dorronsoro, editor, *Proc. of ICANN*, volume 2415 of *LNCS*, pages 999–1004. Springer, 2002.

[142] J. Peltonen, A. Klami, and S. Kaski. Improved learning of Riemannian metrics for exploratory analysis. *Neural Networks*, 17:1087–1100, 2004.

[143] T. Pisanski and J. Shawe-Taylor. Characterizing graph drawings with eigenvectors. *Journal of Chemical Information and Computer Sciences*, 40:567–571, 2000.

[144] M. Quist and G. Yona. Distributional scaling: An algorithm for structure-preserving embedding of metric and nonmetric spaces. *Journal of Machine Learning Research*, 5:399–430, 2004.

[145] M. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proc. of ICML*, 2008.

[146] M. Rattray. A model-based distance for clustering. In *Proc. of IJCNN*, volume 4, 2000.

[147] M. A. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141:660–678, 2002.

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

[149] S. Roweis, L. K. Saul, and G. E. Hinton. Global coordination of local linear models. In *Proc. of NIPS*, volume 14, 2002.

[150] D. E. Rumelhart, G. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[151] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighborhood structure. In *Proc. of AISTATS*, 2007.

[152] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, May 1969.

[153] G. Sanguinetti. Dimensionality reduction of clustered data sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):1–6, Mar 2008.

[154] C. Santa Cruz and J. R. Dorronsoro. A nonlinear discriminant algorithm for data projection and feature extraction. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Proc. of ICANN*, volume 1112 of *LNCS*, pages 563–568. Springer, 1996.

[155] E. Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

[156] A. Schilling. Analyse von Verfahren zur Effizienzsteigerung von multidimensionaler Skalierung. Bachelor's thesis, Bauhaus-Universität Weimar, Fakultät Medien, 2007.

[157] J. J. Schneider and S. Kirkpatrick. *Stochastic optimization*. Springer, 2006.

[158] B. Schölkopf, A. Smola, and K.-R. Müller. Kernel principal components analysis. In *Advances in kernel methods: support vector learning*, pages 327–352. MIT Press, Cambridge, MA, USA, 1999.

[159] L. Scrucca. Dimension reduction for model-based clustering. *Statistics and Computing*, 2009.

[160] M. Seeger. Covariance kernels from Bayesian generative models. In *Proc. of NIPS*, volume 14, pages 905–912, Cambridge, MA, USA, 2002. MIT Press.

[161] F. Sha and L. K. Saul. Analysis and extension of spectral methods for nonlinear dimensionality reduction. In *Proc. of ICML*, 2005.

[162] B. Shaw and T. Jebara. Minimum volume embedding. In *Proc. of AISTATS*, 2007.

[163] B. Shaw and T. Jebara. Structure preserving embedding. In *Proc. of ICML*, 2009.

[164] R. N. Shepard. The analysis of proximities: multidimensional scaling with an unknown distance function. *Psychometrika*, 27(3):219–246, 1962.

[165] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel distributed processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, USA, 1986.

[166] M. Strickert, P. Schneider, J. Keilwagen, T. Villmann, M. Biehl, and B. Hammer. Discriminatory data mapping by matrix-based supervised learning metrics. In L. Prevost, S. Marinai, and F. Schwenker, editors, *Proc. of Artificial Neural Networks in Pattern Recognition (ANNPR)*, volume 5064 of *LNAI*, pages 78–89. Springer, 2008.

[167] M. Strickert and U. Seiffert. Correlation-based data representation. In M. Biehl, B. Hammer, M. Verleysen, and T. Villmann, editors, *Similarity-based Clustering and its Application to Medicine and Biology*, number 07131 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[168] M. Sugiyama. Local Fisher discriminant analysis for supervised dimensionality reduction. In *Proc. of ICML*, pages 905–912, 2006.

[169] Y. W. Teh and S. Roweis. Automatic alignment of local representations. In *Proc. of NIPS*, volume 15, 2003.

[170] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2322, 2000.

[171] S. T. Teoh and K.-L. Ma. RINGS: A technique for visualizing large hierarchies. In S. G. Kobourov and M. T. Goodrich, editors, *Proc. of GD*, pages 268–275. Springer, 2002.

[172] M. E. Tipping. Deriving cluster analytic distance functions from Gaussian mixture models. In D. Willshaw and A. Murray, editors, *Proc. of ICANN*, pages 815–820. IEE Press, 1999.

[173] M. E. Tipping and C. M. Bishop. Probabilistic principal components analysis. *Journal of the Royal Statistical Society B*, 61:611–622, 1999.

[174] W. S. Torgerson. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17(4):401–419, 1952.

[175] W. S. Torgerson. Multidimensional scaling of similarity. *Psychometrika*, 30(4):379–393, 1965.

[176] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig. Accurate telemonitoring of Parkinson's disease progression by non-invasive speech tests. *IEEE Transactions on Biomedical Engineering*, 2009.

[177] K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.-R. Müller. A new discriminitive kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002.

[178] R. Urtasun and T. Darrell. Discriminative Gaussian process latent variable model for classification. In *Proc. of ICML*, 2007.

[179] R. van der Heiden and F. C. A. Groen. The Box-Cox metric for nearest neighbour classification improvement. *Pattern Recognition*, 30(2):273–279, 1997.

[180] L. van der Maaten. Learning a parametric embedding by preserving local structure. In *Proc. of AISTATS*, volume 5 of *JMLR: W&CP*, pages 384–391, 2009.

[181] L. van der Maaten. Preserving local structure in Gaussian process latent variable models. In *Proceedings of Benelearn-09*, pages 81–88, 2009.

[182] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[183] L. van der Maaten, E. Postma, and H. van den Herik. Dimensionality reduction: A comparative review. Technical Report TiCC-TR 2009-005, Tilburg University, 2009.

[184] J. Venna and S. Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proc. of ICANN*, volume 2130 of *LNCS*, pages 485–491, 2001.

[185] J. Venna and S. Kaski. Local multidimensional scaling with controlled tradeoff between trustworthiness and continuity. In *Proc. of WSOM*, 2005.

[186] J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19:889–899, 2006.

[187] J. Venna and S. Kaski. Nonlinear dimensionality reduction as information retrieval. In *Proc. of AISTATS*, 2007.

[188] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *Journal of Machine Learning Research*, 11:451–490, 2010.

[189] J. J. Verbeek, S. Roweis, and N. Vlassis. Non-linear CCA and PCA by alignment of local models. In *Proc. of NIPS*, volume 16, pages 297–304, 2004.

[190] J.-P. Vert, K. Tsuda, and B. Schölkopf. A primer on kernel methods. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, chapter 2, pages 35–70. MIT Press, 2004.

[191] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266, 1997.

[192] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[193] M. Vlachos, C. Domeniconi, D. Gunopulos, G. Kollios, and N. Koudas. Nonlinear dimensionality reduction techniques for classification and visualization. In *Proc. of the 8th ACM SIGKDD (International conference on Knowledge discovery and data mining)*, pages 645–651. ACM, 2002.

[194] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 2007.

[195] Q. Wang and J. Li. Combining local and global information for nonlinear dimensionality reduction. *Neurocomputing*, 72:2235–2241, 2009.

[196] S. Wang and R. Jin. An information geometry approach for distance metric learning. In *Proc. of AISTATS*, pages 591–598, 2009.

[197] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[198] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393:440–442, 1998.

[199] A. R. Webb and D. Lowe. The optimised internal representation of multilayer classifier networks perfoms nonlinear discriminant analysis. *Neural Networks*, 3:367–375, 1990.

[200] K. Q. Weinberger, B. D. Packer, and L. K. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *Proc. of AISTATS*, 2005.

[201] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[202] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, Feb 2009.

[203] K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proc. 21st International Conference on Machine Learning*, pages 839–846, 2004.

[204] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proc. of ICML*, 2008.

[205] C. K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10:1203–1216, 1998.

[206] C. K. I. Williams and M. Seeger. The effect of the input density distribution on kernel-based classifiers. In *Proc. of ICML*, 2000.

[207] M. P. Windham. Parameter modification for clustering criteria. *Journal of Classification*, 4:191–214, 1987.

[208] L. Yang and R. Jin. Distance metric learning: a comprehensive survey. Technical report, Michigan State University, USA, 2006.

[209] L. Yang, R. Jin, R. Sukthankar, and Y. Liu. An efficient algorithm for local distance metric learning. *Proc. of AAAI*, pages 543–548, 2006.

[210] J. Ye, Z. Zhao, and H. Liu. Adaptive distance metric learning for clustering. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2007.

[211] I.-C. Yeh. Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

[212] Y.-R. Yeh, S.-Y. Huang, and Y.-J. Lee. Nonlinear dimension reduction with kernel sliced inverse regression. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1590–1603, 2009.

[213] G. Young and A. S. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3(1):19–22, 1938.

[214] S. Yu, K. Yu, V. Tresp, H.-P. Kriegel, and M. Wu. Supervised probabilistic principal components analysis. In *Proc. of KDD*, pages 464–473, 2006.

[215] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Proc. of NIPS*, 2004.

[216] Y. Zhang and D.-Y. Yeung. Heteroscedastic probabilistic linear discriminant analysis with semi-supervised extension. In *Proc. of ECML PKDD*, volume 5782 of *LNAI*, pages 602–616, 2009.

[217] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. Technical Report CSE-02-019, Pennsylvania State University, CSE, 2002.

[218] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang. Evolutionary extreme learning machine. *Pattern Recognition*, 38:1759–1763, 2005.

Aalto-DD 94/2011

BUSINESS + ECONOMY
ART + DESIGN + ARCHITECTURE
SCIENCE + TECHNOLOGY
CROSSOVER
DOCTORAL DISSERTATIONS

ISBN 978-952-60-4312-8 (pdf)
ISBN 978-952-60-4311-1
ISSN-L 1799-4934
ISSN 1799-4942 (pdf)
ISSN 1799-4934

Aalto University
School of Science
Dept of Biomedical Engineering and Computational Science
www.aalto.fi