Aalto University
School of Science
Degree Programme of Engineering Physics and Mathematics

Mika Juuti

Stochastic Discriminant Analysis

Master's Thesis
Espoo, March 17, 2015

Supervisor: Professor Juha Karhunen
Advisor: Francesco Corona D.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Engineering Physics and Mathematics

ABSTRACT OF
MASTER'S THESIS

Author: Mika Juuti

Title:
Stochastic Discriminant Analysis

Date: March 17, 2015 **Pages:** 85

Major: T-120 Computer Science **Code:** Mat-2

Supervisor: Professor Juha Karhunen

Advisor: Francesco Corona D.Sc. (Tech.)

The processing powers of computers have increased constantly during the last decades. Ordinary personal computers are now able to perform intricate calculations with datasets. Large datasets, such as images, create unique challenges as they often contain more variables than used in ordinary statistical analysis.

In *dimension reduction* we are decreasing the amount of variables by combining them. The locations of the data points in a low-dimensional space are often optimized with respect to some predefined criteria. If we use a response variable to guide the search of the subspace, the method is called a supervised method. When the objective of the dimension reduction is to reduce the size of the space to two or three dimensions, the procedure is often called visualization. This thesis is mostly focused on supervised visualization.

This thesis first discusses a supervised dimension reduction tool developed at the Aalto University: Supervised Distance Preserving Projections (SDPP). The method matches response space distances with linearly transformed input space distances using the Euclidean divergence. Second, this thesis introduces a new method for dimension reduction based on the SDPP: Stochastic Discriminant Analysis (SDA). The method matches point-to-point neighbor probabilities in the linearly transformed input space with target probabilities from the response space, using the relative entropy (Kullback-Leibler divergence). Finally, the performance of this method is analyzed against some selected supervised state-of-the-art dimension reduction methods on contemporary datasets.

Keywords: visualization, linear projections, supervised learning, dimension reduction, Kullback-Leibler divergence

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Mika Juuti		
Työn nimi:	Stokastinen diskriminanttianalyysi		
Päiväys:	17. maaliskuuta 2015	**Sivumäärä:**	85
Pääaine:	T-120 Tietojenkäsittelytiede	**Koodi:**	Mat-2
Valvoja:	Professori Juha Karhunen		
Ohjaaja:	Tohtori Francesco Corona		

Viimeisten vuosikymmenten aikana tietokoneiden prosessointikyky on jatkuvasti kasvanut. Suurten datajoukkojen tutkiminen on tärkeää muun muassa sosiaalisen median alalla. Samalla tavalliset tietokoneet ovat kehittyneet tehokkaiksi prosessointiyksiköiksi, mahdollistaen monimutkaisenkin datan käsittelyn. Toisaalta datajoukot sisältävät usein liian paljon muuttujia havaintojen lukumäärään verrattuna, luoden niin kutsutun dimensionaalisuuden kirouksen.

Uloitteisuuden pienentämisessä pyritään pienentämään muuttujien määrää yhdistelemällä muuttujia. Näissä menetelmissä datapisteiden olinpaikkaa aliavaruudessa optimoidaan usein jonkin kriteerin suhteen. Uloitteisuuden pienentämisessä voidaan käyttää hyväksi havantoihin liittyvää vastetta, jolloin pienentämistä kutsutaan ohjatuksi menetelmäksi. Kun tarkoituksena on pienentää avaruuden kokoa kaksi- tai kolmiuloitteiseksi silmämääräistä tarkastelua varten, pienentämistä kutsutaan visualisoinniksi. Tässä työssä keskitytään ohjattuihin visualisointimenetelmiin.

Tässä diplomityössä tarkastellaan aluksi erästä Aalto Yliopistolla kehitettyä ohjattua uloitteisuuden pienentämismenetelmää: Supervised Distance Preserving Projections (SDPP:tä). Menetelmä sovittaa havaintopisteiden väliset etäisyydet vastepisteiden välisiin etäisyyksiin naapurustossa Euklidista divergenssiä käyttäen. Tämän jälkeen kehitetään SDPP:n pohjalta uloitteisuuden pienentämismenetelmä (Stokastinen Diskriminanttianalyysi, SDA) ja menetelmän käytännöllisyyttä tutkitaan. Menetelmä sovittaa epälineaarisesti muunnetut aliavaruuden etäisyydet havaintopisteiden välillä vastepisteiden välisiin vastikkeisiin relatiivista entropiaa (Kullback-Leibler divergenssiä) käyttäen. Työn lopuksi verrataan menetelmän toimivuutta toisia vastaavia menetelmiä vastaan nykyaikaisilla datajoukoilla.

Asiasanat:	visualisointi, projektio, ohjattu oppiminen, uloitteisuuden pienentäminen, Kullback-Leibler divergenssi
Kieli:	Englanti

Aalto-universitetet
Högskolan för teknikvetenskaper
Utbildningsprogrammet i teknisk fysik och matematik

SAMMANDRAG AV
DIPLOMARBETET

Utfört av:	Mika Juuti		
Arbetets namn:	Stokastisk diskriminantanalys		
Datum:	Den 17 Mars 2015	**Sidantal:**	85
Huvudämne:	T-120 Datavetenskap	**Kod:**	Mat-2
Övervakare:	Professor Juha Karhunen		
Handledare:	Doktor Francesco Corona		

Under de senaste decennierna har moderna bordsdatorer utvecklats avsevärt. Modern processeringsförmåga möjliggör komplicerad databehandling. I analysering av stora datamängder, behöver man dock i praktiken förhandsbehandla datan för att minska minnesanvändningen och processeringstiden.

I *dimensionsförminskning* strävar man efter att minska antalet variabler i datan. Ifall responsvariabler som associeras med observationerna används till nytta kallar man förminskningsmetoderna handledda. Dimensionsförminskning kallas också visualisering då man förminskar datan till två eller tre dimensioner med avsikt att betrakta datan ögonmässigt. I detta diplomarbete fokuserar jag på handledda visualiseringsmetoder.

Först undersöks handledda dimensionsförminskningsmetoden Supervised Distance Preserving Projections (SDPP). Metoden passar ihop avstånd mellan datapunkter i en lågdimensionell projektion av indatan med motsvarande avstånd i utdatan genom att använda Euklidiska divergensen. Sedan utvecklas en motsvarande metod baserad på sannolikhetskalkyl och metodens lämplighet till dimensionsförminskning undersöks. Metoden passar ihop icke-linjärt projiserade avstånd i indatan med motsvarande avstånd i utdatan genom att utnyttja relativa entropin (Kullback-Leibler divergens) mellan de sannolikhetsfördelningar som dessa avstånd ger upphov till. I slutet av diplomarbetet jämförs metodens nyttighet med andra motsvarande metoder.

Nyckelord:	visualisering, projektion, handledd inlärning, dimensionsförminskning, Kullback-Leibler divergens
Språk:	Engelska

Acknowledgements

I wish to thank my adviser Francesco Corona for his help in my thesis. The discussions with Francesco were an immense help in navigating through the research work and arriving at this thesis. His constant positive attitude and helpful remarks were ever so important. I also want to thank professor Juha Karhunen for the possibility to work in the Information and Computer Science department on the Master's thesis and for extending my work period until March 2015. I am very grateful for the opportunity to work as a summer intern at the Information and Computer Science department. Thank you, and keep up the good work!

Finally, I wish to thank my family and my lovely girlfriend for their support.

Espoo, March 17, 2015

Mika Juuti

Contents

Chapter 1

Introduction

1.1 Data

The last decades have marked an explosive growth in the amount of data all over the globe. Hyperspectral imagery, computational finance, social network analysis and biological micro-array analysis amongst others contain a very large amount of recorded variables [19, 20, 36]. Ideally, we would want to make use of the full amount of data available. But in practice, the full use of these kinds of datasets is and will be both too much for the memory and too slow to process. The processing speed is frequently linked to the size of the data matrices representing the data.

Dimension reduction is related to this fundamental setting. Which portion of the data is useful and which can be discarded? Often the answer to the question is related to how we want to use the data. If we do dimension reduction utilizing some response variable we are interested in, then we are doing *supervised* dimension reduction. We often try to squeeze out as much information in as a small space as we can.

1.2 Data Analysis

Conventional statistical methods are designed to analyze a few well selected variables with a large amount of observations. Data analysis [19] methods usually deal with the opposite phenomena: there is an abundance of variables and the amount of observations varies dramatically.

In the data analysis setting, data matrices are frequently thought to represent points in a high-dimensional space. To give a concrete example, let us think that we have data of different countries in the EU. Each country has a corresponding high-dimensional point or data vector. Each dimension represents a variable associated with that country, e.g. the average income or the highest score ever reached in the Eurovision song contest. Frequently, the data dimensionality far exceeds three dimensions, and it is therefore often difficult to imagine what the data looks like geometrically.

Three of the most common data analysis tasks are *classification*, *regression* and *clustering* [4]. Quite often one of the values in the data matrix will be estimated from other variables y_i in the matrix. In the case that the estimated variable y_i is a categorial variable (yes/no, male/female, car/boat/train,...) the task of estimating this variable is called a classification task. In the case that the variable y_i takes real values the task is called

regression [45]. For instance, in linear regression the response variable y is modeled as if it has been generated by the model $y = \mathbf{w}^T\mathbf{x} + \epsilon$, where $\mathbf{w} \in R^{D\times 1}$ denotes the weight vector for the variables $\mathbf{x} \in R^{D\times 1}$ and ϵ is (additive) noise [45]. Regression problems can be written in the form $y = f(\mathbf{w}, \mathbf{x}) + \epsilon$, where f can be a nonlinear function. Cluster analysis, or clustering, is the task of grouping the elements in the data set in such a way that similar objects are close to each other [19].

1.3 Dimension Reduction

Dimension reduction is a crucial preprocessing step in machine learning, after which classification, regression or cluster analysis is often done. Historically, dimension reduction began with the development of principal component analysis (PCA) [4, 33, 37, 59] in the early 20th century. PCA was discovered by Pearson in 1901 and was later independently studied by Hotelling in 1933. However, multivariate analysis only really became popular after the development of the electronic computers in the latter half of the 20th century. Today, dimension reduction (DR) tools are included in virtually every statistical software package available online, in the form of PCA and related methods.

In machine learning literature, many subfields have developed their own terms for the same phenomena. In pattern recognition literature the term *feature* is preferred over *variable* [29]. Sometimes a distinction is made such that processed variables are called features, but often these two terms are used interchangeably. Dimension reduction is usually separated into feature extraction and feature selection [4]. *Feature selection* concerns the selection of a subset of variables. Methods in feature selection include forward selection and backward selection. *Feature extraction* concerns transforming variables into new features either non-linearly or linearly (by matrix multiplication). For example, PCA is a linear feature extraction method. To clarify, in this thesis, we are referring to feature extraction, whenever we talk about dimension reduction.

In data analysis applications, the purpose is often to create a predictor model, which can predict the class label or regression value of a new observation [27]. Dimension reduction reduces the amount of variables that are processed, making the model simpler. A simpler model is more robust on small data sets because it contains less variance, making it less dependent on the particulars in the data set, including noise and outliers [4]. If the data can be represented without loss of information in a few dimensions we can visualize the dataset (embedding it to two or three dimensions) to see its structure and eventual outliers [4].

Dimension reduction has two purposes: we can either search for a good low-dimensional linear projection or try to find a low-dimensional geometrical shape embedded in a high-dimensional space. The former case includes PCA, Locality Preserving Projection (LPP) [31] and Supervised Distance Preserving Projections (SDPP) [66]. The latter case is often called manifold learning. Some famous manifold learning algorithms are Isomap [4], Locally Linear Embedding (LLE) [4], Laplacian Eigenmaps [12] and recently, Stochastic Neighbor Embedding (SNE) [32] and its varieties.

One way to further differentiate between dimension reduction methods is whether output information is used. Linear Discriminant Analysis (LDA) [4] is one of the best known supervised dimension reduction methods based on class information. Methods such as SDPP work for both regression and classification settings. In recent years Stochastic

Neighbor Embedding has risen to one of the top exploratory dimension reduction tools available. Motivated by the success of t-distributed SNE (t-SNE) [61] at visualizing various data sets, we search for methods of how to combine the knowledge of t-SNE with SDPP.

1.4 Contributions of this thesis

This thesis interprets the SDPP and shows how the SDPP works. Then a new method called Stochastic Discriminant Analysis is presented. It preserves local neighborhoods in the output space by minimizing an information theoretical quantity, the Kullback-Leibler divergence. The new method is especially suited to project multiclass data into a low dimensional embedding. This internship also included other work related to the SDPP, which is summarized in Section 3.5.

Chapter 2 introduces the necessary background knowledge for understanding this thesis. Section 2.1 introduces dimension reduction and the curse of dimensionality. Section 2.2 introduces different dimension reduction methods, including the PCA and LDA. Section 2.3 introduces machine learning concepts, including the bias-variance trade-off and regularization. Section 2.4 introduces some ideas used in nonlinear mathematical optimization and introduces contemporary methods for solving nonlinear optimization problems. Section 2.5 introduces information theoretical concepts used in this thesis, mainly entropy and the Kullback-Leibler divergence. Chapter 3 introduces the dimension reduction tool SDPP and shows how to calculate the projection matrix of it.

Chapter 4 introduces the new dimension reduction tool SDA and discusses how to find the projection matrix efficiently. Section 4.1 analyzes the cost function and shows a regular geometrical structure that is searched. The derivation of the gradient of the objective function is discussed and the computational complexity of the method is determined in Section 4.2. Section 4.3 discusses the Hessian of the objective function.

Chapter 5 introduces the datasets used in the experimental evaluation section, with some examples of images in the image datasets. Chapter 6 evaluates the dimension reduction technique SDA experimentally. Section 6.1 discusses the optimization environment used (Matlab) and the third-party optimization packages used. Sections 6.2, 6.3 and 6.4 contain an in-depth analysis of three contemporary datasets (Olivetti, USPS and COIL-20). The 1-$\mathcal{NN}$ classification accuracy calculated from the data different contemporary dimension reduction methods on a range of target dimensions is calculated and their two-dimensional embeddings are shown. The runtime of SDA was analyzed and efficient choices for optimization algorithms were found in Section 6.5. Section 6.6 evaluates two-dimensional projections of the dataset in terms of classification accuracy of out-of-sample projection of test points, evaluated on six additional datasets. Chapter 7 summarizes the contributions of this thesis.

Chapter 2

Background

In this chapter, the relevant background knowledge for understanding this thesis is presented. The concepts of *dimension reduction* and the *curse of dimensionality* [19] are discussed. An important topic in machine learning; the bias-variance trade-off, is introduced, as well as some dimension reduction methods with examples. Then, the necessary optimization background is discussed. This is mainly related to the concept of optimality and the various ways of minimizing a function using gradient information. Finally, some relevant information theoretical concepts are discussed with examples, mainly the entropy and the Kullback-Leibler divergence.

2.1 The curse of dimensionality

The curse of dimensionality [19] is often the motivation for doing dimension reduction. The curse of dimensionality refers to the many phenomena that are associated with high-dimensional data, but are concealed when dealing with low-dimensional data. The term originates from Bellman [19], from his analysis of dynamic systems. In this thesis, high-dimensional data refers to mathematical vector spaces with dimensions much higher than three. The vectors can be manipulated like any other mathematical vector: multiplications by scalars, vectors and matrices can be done and distances between vectors can be calculated just as in the low-dimensional variants. The values of the vector can also be thought to represent a point in the high-dimensional space in the same manner that two-dimensional vectors represent points on a plane and three-dimensional vectors represent points in space. Planes in higher than two dimensions are often called hyperplanes and spheres in higher dimensions are called hyperspheres. Some surprising differences between low-dimensional space and high-dimensional sample space are highlighted next.

High-dimensional data is sparse

The volume of the empty space between data points rises exponentially for each added dimension, while the volume of the occupied space rises linearly for each added sample. Let us think of uniformly distributed data in the unit hypercube $[0, 1]^{10}$. Let us assume that we have 100 points in 10 dimensions. We draw a hypercube around each point, with the side lengths 0.1, 10% of the side length in the unit hypercube. We can assume for simplicity that the small cubes do not overlap. In that case, the boxes around the

points occupy a volume of $100 \cdot 0.1^{10} = 10^{-8}$, compared to the total space volume 1. To occupy the full volume of the $10D$ space, 10^{10} boxes would be needed. In a 20D space, the proportion would be 10^{-18}. To occupy the full $20D$ volume of the space, 10^{20} boxes would be needed. What seemed like a lot of points in a low-dimensional space actually only occupies a tiny fraction of the high-dimensional space. A high-dimensional space is inherently *sparse*. This makes it often possible to find meaningful linear subspaces in the data.

Distances between points lose meaning

When working with real and integer data, small differences in an individual variable's value can result in unseemly large distances between data points. This phenomena occurs frequently with so called bag-of-words datasets, where each word in a defined dictionary (for example the whole Finnish dictionary or a subset of it) is a (variable) dimension. Take for example data points (vectors) x_1 and x_2 and assume that they represent the same document, however a word is added to x_2, which is not occurring in x_1. This causes the attribute $x_{1i} = 0$ to change to value $x_{2i} = 1$. The distance between these two data sets changes from $|x_1 - x_2| = 0$ to $|x_1 - x_2| = \sqrt{1} = 1$. In the case that two words are different, the distance becomes $|x_1 - x_2| = \sqrt{2}$ and so on. If one document contains two occurences of a word not occurring in the other document, the distance becomes $|x_1 - x_2| = 2$. As such, the distances between points become quantized. The distance value does not reflect our idea of how similar these documents are! Even so, nearest neighbors calculated from distances are still useful.

Irrelevant variables

The problem of irrelevant variables is related to the previous problem. Imagine that we have gathered a large data set with different attributes. Each of these variables would constitute a dimension of its own. If we want to predict a specific phenomena from the data (e.g. how far a subject is able to jump), certain variables might only add to the confusion of how far the subject jumps. For example, the skill in Finnish language or the price of the subject's shirt should hardly affect the performance, although they might end up correlating on some datasets. These variables do, however, affect the perception of adjacency in the point space. Irrelevant variables contain irrelevant noise to the data set. Distances between samples get additive noise, which makes it difficult to draw conclusion based on distances between points. Often it is precisely these distances between points that are used to cluster data points.

Redundant variables

Similarly, we can imagine situations where there are near duplicate variables in the dataset. Often it is better to discard one of the highly correlating variables or to combine them, as they often would convey the same underlying information about the data.

2.2 Dimension reduction methods

The previous section presented problems that high-dimensional data creates for data analysis. Shlens calls dimension reduction the process of finding important axes in a data set and dropping off uninteresting axes [59]. This section introduces different dimension reduction ideas. History of dimension reduction is described and some recent advances are discussed.

2.2.1 Principal Component Analysis

The invention of the earliest dimension reduction method is often accredited to Pearson, for the discovery of Principal Component Analysis in 1901. In his paper "On lines and planes of closest fit to systems of points in space" [49], Pearson was searching for best fitting lines or planes through three or higher dimensional spaces. He minimized the sum of the squared distances between the data points and the closest point on a line, see Figure 2.1. Among other things, Pearson concluded that "the best fit for a line of any order goes through the centroid of the system", i.e. the center of mass for all points. In 1933, Hotelling independently improved on the method in the paper "Analysis of a complex of statistical variables into principal components" [33]. Hotelling was examining scores in mental tests and wanted to find a new set of *factors*, or as he preferred to call it *components*, of the scores that were orthogonal and contained the maximum amount of variability of the data. Orthogonality means that the inner product of the vectors is zero and consequently that they are uncorrelated. The new variables were ordered in such a way that the first *principal components of variance* retain most of the variation among all the original variables. Additionally, the components with a small contribution to the total variance were discarded. Hotelling explained that geometrically, the method corresponded to rotating the coordinate axes to ellipsoids of uniform density, see the simplistic example in Figure 2.2. Today, dimension reduction with PCA is probably the first dimension reduction method to test on any given data set due to its simplicity and ease of interpretation.

Generally, the principal components for a data matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$ with n data vectors and D variables are found by its covariance matrix $\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$, where $\mathbf{\Sigma} \in \mathbb{R}^{D \times D}$ is the covariance matrix of the data matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$. It is assumed that the data has zero mean. If the mean is not zero, it can be subtracted from the data vectors making them mean zero. $\mathbf{W} = [\mathbf{W}_1\mathbf{W}_2 \dots \mathbf{W}_d] \in \mathbb{R}^{D \times D}$ are the d projection vectors and $\mathbf{\Lambda}_{ii}$ are the eigenvalues λ_i $(i = 1, \dots, d)$ of the covariance matrix $\mathbf{\Sigma}$ arranged on the diagonal in descending order. In general, a subset of the principal components are selected so that enough of the variance is explained, i.e. $(\sum_i^k \lambda_k)/(\sum_i^d \lambda_i) \geq \alpha$, where α is some fractional threshold (e.g. 0.9).

The projection matrix $\hat{\mathbf{W}} = [\mathbf{W}_1 \dots \mathbf{W}_k]$ is formed using the eigenvectors associated with these eigenvalues. The low dimensional projection is then $\mathbf{Z} = \mathbf{X}\hat{\mathbf{W}} \in \mathbb{R}^{n \times k}$. PCA can be derived from many optimization criteria, such as the maximization of variance or the minimization of mean-square error. PCA can also be derived from a probabilistic model, where the variables are assumed to be Gaussianly distributed. PCA extacts optimal variables for Gaussianly distributed data [16, 45].

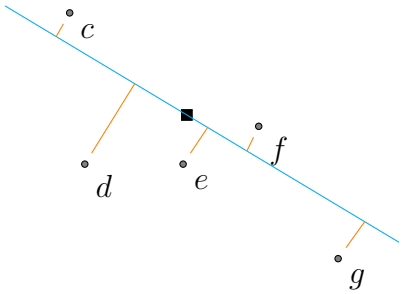

Figure 2.1: A schematic of the intuition behind PCA. The line that minimizes the sum of the squared distances between the blue line and the points $\{c, d, e, f, g\}$ is called the first principal component. Note that the principal component goes through the unlabeled center of mass, the centroid, of the points.

2.2.2 Singular Value Decomposition

Principal Component Analysis is related to the Singular Value Decomposition (SVD) [6, 43, 59]. In general, any $m \times n$ matrix $\mathbf{A}$ is decomposed as

$$\mathbf{A} = \mathbf{USV}^T, \tag{2.1}$$

where $\mathbf{S} \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing the singular values. $\mathbf{U} \in \mathbb{R}^{m \times m}$ is a matrix containing the *left*-singular vectors and $\mathbf{V}^T \in \mathbb{R}^{n \times n}$ contains the *right*-singular vectors. The left-singular vectors and right-singular vectors are orthonormal. SVD can be used to calculate the principal components by writing $\mathbf{X}^T\mathbf{X} = (\mathbf{USV}^T)^T(\mathbf{USV}^T) = \mathbf{VS^2V}^T$. Here $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ (orthonormality) and we can see that the diagonal matrix containing the eigenvalues in PCA is $\mathbf{\Lambda} = \mathbf{S}^2$ and the PCA transformation matrix is $\mathbf{W} = \mathbf{V}$. The singular values are the non-negative square roots of the eigenvalues. For square matrices $\mathbf{A}$, the singular vectors are the eigenvectors, where the left-singular vectors are the same as the right-singular vectors: $\mathbf{U} = \mathbf{V}$. PCA is in fact SVD applied to the covariance matrix $\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X}$. SVD is a generalization of PCA for non-square matrices [30]. SVD can be calculated from the matrix $\mathbf{AA}^T$ or matrix $\mathbf{A}^T\mathbf{A}$. It is computationally efficient to choose the matrix with a smaller dimensionality.

When the input matrix has far more rows m than columns n the matrix is called *thin*. In this case, a computationally efficient version of SVD can be calculated [45]. Instead of calculating the full SVD, only the n first columns of matrix $\mathbf{U}$ are calculated and only the n first rows of the diagonal matrix $\mathbf{S}$ are calculated: $\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{S}}\mathbf{V}$. The motivation is that the last rows in matrix $\mathbf{S}$ are zeros and as such any vector in $\mathbf{U}$ that is multiplied by these rows have no influence. Their calculation is skipped. The SVD obtained this way is called the thin SVD or the economy sized SVD [45]. The calculations can be similarly speeded up when there are more columns than rows [45].

2.2.3 Random projections

The motivation for random projection [15] stems from the Johnsson-Lindenstrauss lemma, namely that distances are approximately preserved if the embedding dimension is high enough. The weights for the elements in the projection matrix are generated from any

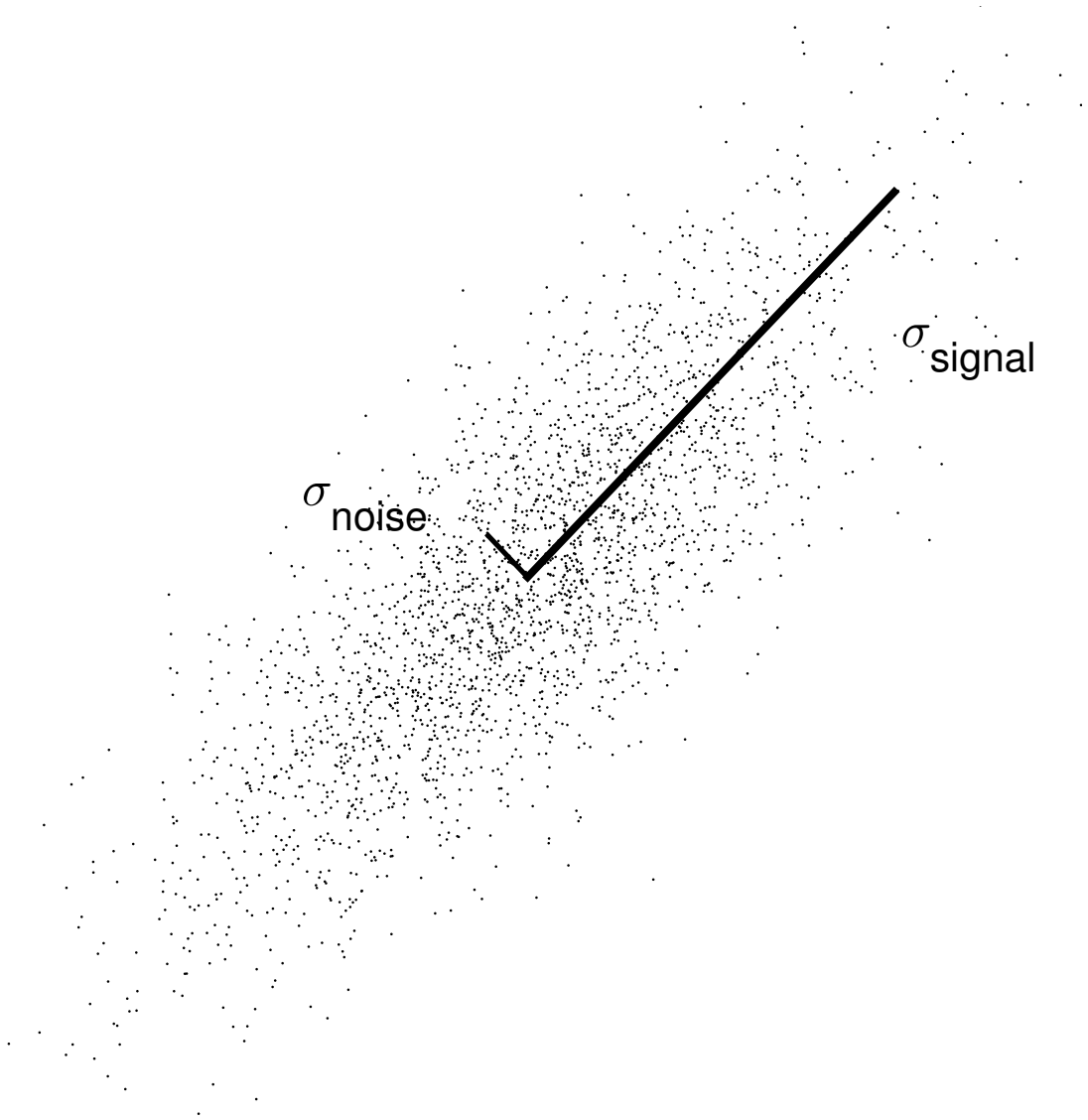

Figure 2.2: PCA projects the data in the directions containing most of the variability in the data. Most of the data variation can be found along σ_{signal} in this picture. The second principal component σ_{noise} is orthogonal to the first.

probability distribution with the expected value zero. Random projections are used in Extreme Learning Machines [34, 44].

2.2.4 Fisher's Linear Discriminant Analysis

A method for *supervised* dimension reduction was published a few years after PCA. In 1936, R.A. Fisher investigated the nowadays famous *Iris* dataset [21], that contains 150 flowers from three very similar species, each with measurements of the petal width, petal length, sepal width and sepal length. The method that Fisher developed is nowadays called the Fisher Discriminant Analysis. It finds a subspace, where different classes are separated from each other in an optimal way according to a criterion. Although Fisher originally was investigating only two different classes at one time, the method was extended to multiple classes by Rao in 1948 [35, 51]. Fisher's Linear Discriminant Analysis (FDA or LDA) is still today used extensively as it is a simple method that has reasonable performance in many cases.

The idea in the FDA is that we want to maximize the distance between classes while minimizing the distances within classes. This is approached by maximizing the FDA criterion in Equation (2.2) [4]

$$\max_{\mathbf{W}} J(\mathbf{W}) = \frac{\mathbf{W}^T \mathbf{S}_B \mathbf{W}}{\mathbf{W}^T \mathbf{S}_W \mathbf{W}} \tag{2.2}$$

where $\mathbf{S}_B$ refers to the scatter matrix between classes and $\mathbf{S}_W$ to the scatter matrix within classes. The between class scatter matrix is calculated as $\mathbf{S}_B = \sum_{i=1}^{K} n_i(\mathbf{m}_i - \mathbf{m})^T(\mathbf{m}_i - \mathbf{m})$, where $\mathbf{m}_i \in \mathbb{R}^{1\times D}$ is the mean of the ith class, K is the number of classes in the dataset, $\mathbf{m} = \sum_{i=1}^{K} \mathbf{m}_i/K$ and n_i is the number of observations in class i. The scatter within classes is calculated as $\mathbf{S}_W = \sum_{k=1}^{K} \sum_{i,K_i=k} (\mathbf{x}_i - \mathbf{m}_k)^T(\mathbf{x}_i - \mathbf{m}_k)$. For a demonstration of FDA on a toy problem, see Figure (2.3).

A downside of FDA is that in practice, feature selection has to be done before the usage of FDA. The inversion of the matrix in Equation (2.2) is particularly prohibitive in the analysis of high-dimensional data. FDA is often performed after PCA projection, to increase relevant feature extraction [64].

Eigen- and fisherfaces

The so called eigenfaces and fisherfaces [11] are well-known in face recognition tasks. Eigenfaces are obtained with a PCA projection of the original space. Fisherfaces are obtained by projecting the input space with the subspace obtained with Fisher's Linear Discriminant. Usually, the projection is obtained by first projecting the data with PCA and applying FDA after this preprocessing step to avoid singular values in the matrix inversion. In both methods, it is thought that the faces are separable in some subspace of the original high-dimensional pixel space. Eigenfaces are optimal in the manner that they preserve maximum variance between datapoints, making it easier to differentiate points. But eigenfaces are typically susceptible to lighting conditions and face rotations, thus ending up smearing different classes together [11]. Fisherfaces are more robust to these changes. However, fisherfaces usually requires ν-1 dimensions to differentiate classes, where ν is the number of classes in the data set. The methods are called faces because the

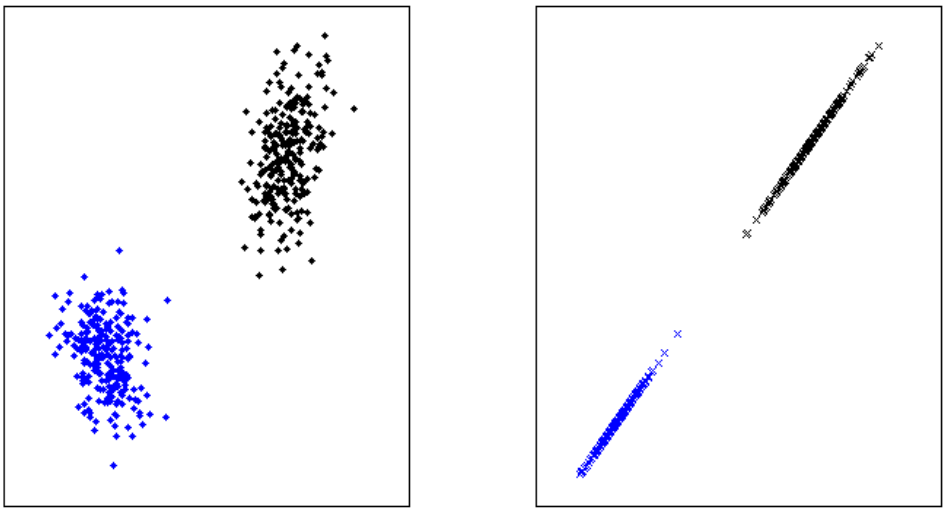

Figure 2.3: In Fisher Discriminant Analysis, the distance between class means are maximized, while the within-class distance to the mean is minimized. The right-side image shows the optimal 1D subspace obtained with the FDA criterion.

subspace directions that they create can be reprojected into the original high-dimensional space (images) and reshaped to faces. The reprojected vectors represent information about either maximum variance in the images (eigenfaces) or strong discrimination ability (fisherfaces). In order for pixel-intensity-based face recognition to work, the images need to be cropped to the same size and centered around the same area, typically the nose and eyes lying on the same position. The eigenfaces and fisherfaces give an intuitive feeling of what pixel values the subspaces weight. For example, if there are d eigenfaces (a d-dimensional problem), a person's face could be composed of (20% eigenface 1) + (53% eigenface 2) + ... + (−3% eigenface d).

2.2.5 Laplacian Eigenmaps

Belkin and Niyogi presented in 2001 a nonlinear manifold learning technique for projecting high-dimensional data into a low-dimensional space in such a way that local points in the high dimensional space are kept close in the projection [12]. The motivation is the assumption that high dimensional data points have an intrinsic low dimensional representation. This low dimensional representation is found by examining the local geometry of the data points. The basic algorithm has two steps:

1. Find the k nearest neighbors for the n D-dimensional data points and assume their matrix form $\mathbf{G} \in \mathbb{R}^{n \times n}$, with n being the number of samples. The matrix $\mathbf{G}$ contains indicator variables: $\mathbf{G}_{ij} = 1$ implies that j is one of the k nearest data points to i.

2. Solve a generalized eigenvalue problem $\mathbf{LZ} = \lambda\mathbf{DZ}$, where $\mathbf{L} = \mathbf{D} - \mathbf{G}$ is the graph Laplacian of $\mathbf{G}$ and $\mathbf{D} \in \mathbb{R}^{n \times n}$ is the degree matrix of $\mathbf{G}$ and $\mathbf{Z} \in \mathbb{R}^{n \times D}$ is the low-dimensional embedding. The degree matrix $\mathbf{D}$ is a diagonal matrix containing the row-sums of $\mathbf{G}$.

Laplacian eigenmaps are able to find structure in high dimensional data, but they depend entirely on the specification of a graph matrix $\mathbf{G}$. The embedded data points $\mathbf{Z} \in \mathbb{R}^{n \times d}$ depend on the original data points through some function, which can be approximated by doing a Nyström approximation of the eigenfunction [13]. The method of computing the Laplacian Eigenmaps by solving the generalized eigenvalue problem is straightforward and the method enjoys some popularity. The Spectral Clustering algorithm embeds the data points into a low-dimensional space and then clusters the data points. The low-dimensional space is obtained with Laplacian Eigenmaps.

2.2.6 Locality Preserving Projections

In 2003, He and Niyogi presented a linear variant of Laplacian Eigenmaps that projected the data points using a linear transformation of the input points [31]. The benefits of using a linear transformation matrix is that the projection is not only defined for the training data points, but in the whole ambient space. This is because a transformation matrix, rather than point-wise mapping, is learned. The basic Locality Preserving Projection (LPP) algorithm is defined similarly as:

1. Find the k nearest neighbors for the data points and assume its matrix form $\mathbf{G}$.
2. Solve a generalized eigenvalue problem $\mathbf{XLX}^T\mathbf{W} = \lambda\mathbf{XDX}^T\mathbf{W}$, where L is the graph Laplacian of $\mathbf{G}$, $\mathbf{D}$ is the degree matrix of $\mathbf{G}$ and $\mathbf{W}$ is the transformation matrix from $\mathbf{X}$ space to $\mathbf{Z} = \mathbf{W}^T\mathbf{X}$ space.

Projections found with Locality Preserving Projections mimic the ones found with Laplacian Eigenmaps. The projection could also be found by first-order optimization methods using the gradient $\mathbf{XLX}^T\mathbf{W}$, however He and Niyogi prefer to use a generalized eigenvalue formulation [31].

2.2.7 Stochastic Neighborhood Embedding

In 2002, Hinton and Roweis published the Stochastic Neighborhood Embedding (SNE) algorithm that minimizes the Kullback-Leibler divergence, a dissimilarity measure between two probability distributions [32]. The method is a nonlinear mapping (manifold learner) of points in a high dimensional space to a low dimensional space. Distances in the input space are modeled as probabilities by transforming them by a Gaussian kernel in the input space:

$$p_{j|i} = \frac{\exp(-\frac{||x_i - x_j||^2}{2\sigma_i^2})}{\sum_{k=1}^{n} \exp(-\frac{||x_i - x_k||^2}{2\sigma_i^2})}. \tag{2.3}$$

The Gaussian kernel is computed similarly in the output space:

$$q_{j|i} = \frac{\exp(-||x_i - x_j||^2)}{\sum_{k=1}^{n} \exp(-||x_i - x_k||^2)}.$$

The Kullback-Leibler divergence of approximating p with q needs to be minimized:

$$C_{SNE} = \sum_i KL(P_i||Q_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{j|i} \log(\frac{p_{j|i}}{q_{j|i}})$$

SNE can find good low dimensional representations of the input space, however it requires the specification of the variance of each data point in the Gaussian kernels in Equation (2.3). Modern implementations of SNE are fairly fast. The mapping function from the input space to the output space is unknown and the learned representation is not generalized to new data points. The problem contains numerous local minima, meaning that different initializations can yield significantly different visualizations.

2.2.8 t-distributed SNE

The t-SNE was proposed by van der Maaten and Hinton in 2007 [61]. It contains two modification compared to SNE: normalization is calculated by dividing with all elements and the distribution of the embedding space is changed to a distribution with a heavier tail. The authors chose Student's t-distribution with one degree of freedom:

$$q_{i,j} = \frac{\frac{1}{1+||x_i - x_j||^2}}{\sum_{k=1}^{n}\sum_{j=1}^{n}\frac{1}{1+||x_k - x_l||^2}}$$

The probabilities calculated from the high-dimensional space are symmetrized versions of the probabilities calculated in SNE Equation (2.3):

$$p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2.4}$$

The cost function in t-SNE is the Kullback-Leibler divergence is

$$C_{t-SNE} = KL(P||Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{i,j} \log(\frac{p_{i,j}}{q_{i,j}})$$

t-SNE is tailored for information retrieval from high-dimensional space and it has been optimized to work for tens of thousands of data elements. Yet the mapping of data points in t-SNE is not obtained in the process and new data points are difficult to place into the projection space. Yang et. al. [65], used Locality-Constrained Linear Coding to interpolate new data points into the mapping. t-SNE is tailored for information retrieval from high-dimensional space.

2.2.9 Other supervised methods

There are some popular linear supervised dimension reduction tools too. The methods used in this thesis are briefly introduced here. *Partial Least Squares* (PLS) regression is a supervised linear dimension reduction technique that tries to find subspaces in the input matrix that explain the largest amount of variance in the response matrix. When used with categorial response variables it is referred to as PLS-DA [50]. *Kernel Dimension Reduction* (KDR) [25] is a sufficient dimension reduction method [3] for classification and regression data. A sufficient dimension reduction contains all the regression information

that the original space contained about the response variable. KDR tries to find the central subspace [3] for the input data, which is the intersection of all dimension reduction subspaces. The method is demanding in terms of computation and memory consumption. A gradient version of KDR has been developed for faster computation, called gKDR [26]. *Supervised PCA* by Barshan et al. [9] is a regression technique that finds the principal components with maximum dependence on the given response variable. SPCA tries to find variables that are orthogonal in a kernel space of the response variable. A dual-space and kernel variant of SPCA (KSPCA) [25] has also been developed, extending the usage scenarios of the method.

2.3 Machine Learning Concepts

In this subsection, the concepts of overfitting and generalization ability will be presented. Strategies of improving the generalization ability of machine learning methods are also discussed.

2.3.1 Bias-variance trade-off

Machine learning methods are methods for learning patterns from supplied data. Machine learning has been applied to variety of areas, amongst others currency trading. Currency trading algorithms can make future predictions based on past history. While the algorithms are remarkable at extracting patterns from the data, it is not possible to learn cause-and-effect relations that are not present in the data. One example of this would be a financial meltdown. Phenomena that do not appear in the training data cannot be predicted. This is called the *zero counting* or *sparse data problem* [45]. This is analogous to a philosophical problem called the *black swan paradox*. It is based on the ancient belief in Europe that all swans are white, and therefore black swans could not exist. Settlers in Australia did however discover black swans, creating the paradox [45]. This kind of overconfidence in the structure of the data is related to *overfitting*, which will be discussed in the next few paragraphs.

The regression problem of estimating the response variable y with the model $y = f(w, x) + \epsilon$ is used often to clarify the difference between the terms *bias* and *variance*. When minimizing the mean square loss in the regression model, a bias-variance decomposition of the error term ϵ can be made. The error term ϵ of the model can be written in the form $\epsilon = \mathbb{E}\Big[\Big(f(\hat{w}, x) - f(\bar{w}, x)\Big)^2\Big] + \Big(f(\bar{w}, x) - f(w^*, x)\Big)^2 = \mathrm{Var}(\hat{w}) + \mathrm{Bias}(\hat{w})^2$. Here $\mathbb{E}$ denotes the expectation, w^* the true value of the variable, $\hat{w}$ the estimated value of the variable and $\bar{w}$ the mean value [45]. Bias refers to part of the error of estimating the true mean, whereas variance refers to part of the error stemming from the spread around estimated mean [29]. The bias-variance trade-off means that it might be beneficial to use a biased estimate, if we can decrease the variance by doing so [45].

Going back to the previous example: using a too simple model increases the currency trading model's bias, while increasing the complexity of a model might decreases its bias but increases the variance. A model which has too much variance might be *overfitting* the data, while a model which has too much bias might be *underfitting* the data, see Figure (2.4). The curve goes through each black learning point, meaning that the residual

(fitting error) $\epsilon = y - \hat{y} = 0$, but the predicting performance of the method is bad. Large variable coefficients in linear methods typically characterize overfitting. The predicted values for the orange square points are fairly far from the actual values. A line has bigger residuals in the training set, but the predictive performance of the method is much better on the orange data points. The ability to predict new values outside of the training set is called *generalization* ability. Amongst others, the Akaike information criterion and the Bayesian information criterion [45] are used to choose less complex models in the case that multiple models fit the data.

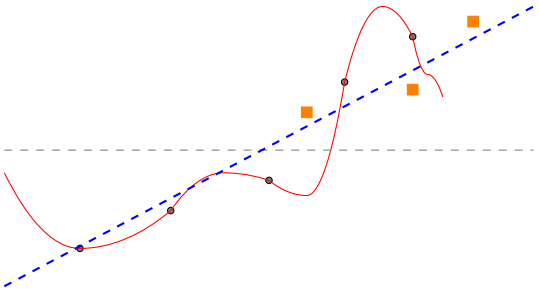

Figure 2.4: An example of overfitting. Applying a very complicated model to a data set reduces the fitting error but increases the validation set error. A very simple model generalizes better to new out-of-sample data. A straight horizontal line (the average value) would be underfitting the structure apparent in the data.

The bias-variance trade-off can be made for classification settings with 0-1 loss too, but error is no longer expressible as a summation of a variance term and a bias term. Instead, the terms are associated multiplicatively [45]. Murphy [45] suggests that instead of focusing of the bias-variance trade-off in classification settings, it is better to focus on the expected loss. The approximation of expected loss, the empirical loss, can be measured with cross-validation.

Computational models often involve some free parameters that must be chosen. The choice of the free parameters can be done by validating the generalization performance of the algorithm over a separate *validation set*. To evaluate the performance of the algorithm, the algorithm is run one more time and the performance is evaluated over a separate *test set*. At this final point the *learning set* and validation set is usually merged. The straightforward way to do this is to split the data in separate parts, e.g. 60% of the data in the learning set, 20% in the validation set and 20% in the test set. This is called hold-out validation. In this thesis, the free parameter is often related to the size of the local neighborhood k.

2.3.2 Regularization

A fairly popular way to decrease a model's variance is to employ the *Tikhonov regularization* (L_2 -regularization or *weight decay*). The original cost function $J(\mathbf{w})$, $\mathbf{w} \in \mathbb{R}^{\nu \times 1}$, ($\nu$ being the number of variables) is supplemented with a bias-term $||\mathbf{\Gamma}(\mathbf{w} - \mathbf{w}_0)||_2^2$, where $\mathbf{w}_0$ is the initial guess (often zeros) and $\mathbf{\Gamma} \in \mathbb{R}^{\nu \times \nu}$ is some appropriate matrix.

$$J_{reg}(\mathbf{w}) = J(\mathbf{w}) + ||\mathbf{\Gamma}(\mathbf{w} - \mathbf{w}_0)||_2^2. \tag{2.5}$$

Most often $\mathbf{\Gamma}$ is chosen to be λI, with an appropriately chosen scalar value λ. In its simplest form the regularization can be expressed as $\lambda \mathbf{w}^T \mathbf{w}$. By penalizing the parameters $\mathbf{w}$, we can assure that the model is simple [45]. Tikhonov regularization is C^1 differentiable and does not require specialized optimization methods, much unlike the other popular form of regularization, i.e. L_1-regularization (LASSO-regularization).

2.3.3 k-$\mathcal{NN}$ classification

The generalization performance in classification settings is often evaluated by classifying data points from a separate test set. The method of the k-nearest neighbors is simple, but nonetheless it is a quite powerful classifier. In it, the k nearest points inside a training set are searched for a test point and majority vote on the class labels is done based on those points. The proportion of points that are correctly classified is the k-$\mathcal{NN}$ classification accuracy. The 1-$\mathcal{NN}$ classifier is the simplest form of k-$\mathcal{NN}$ classification. Here, the first nearest data point for each test point is searched from the training set. The class labels between the test point and the training set point are compared. The procedure is repeated for each test point.

2.3.4 Kernel Trick

The kernel trick is computational procedure that makes it possible to incorporate non-linear pattern recognition to many linear methods. The problem is first transformed into dual space, in which the data is no longer computationally dependent on the primary space, but on the number of samples. Then, the inner product of two data points can be replaced by a suitable nonlinear variant, for example a Gaussian kernel or polynomial kernel, to give nonlinearity to the method. Any kernel satisfying Mercer's theorem [38] can be used in place of the inner product.

2.4 Mathematical Optimization

In this section, relevant mathematical optimization methods are discussed. Drawbacks and strengths of various optimization methods will be presented.

2.4.1 Optimality

When minimizing an objective function $J(\mathbf{x})$, a solution $\mathbf{x}^* \in \mathbb{R}^{n\times 1}$ is said to be *optimal* if the objective function for the solution $J(\mathbf{x}^*)$ is smaller than the objective function evaluated with any other parameter $\mathbf{x} \in \mathbb{R}^{n\times 1}$: $J(\mathbf{x}^*) \leq J(\mathbf{x})$. The solution is said to be a *local optimum* if the $J(\mathbf{x}^*)$ is smaller than any other solution $\mathbf{x}$ inside some region around $\mathbf{x}^*$. If $J(\mathbf{x}^*)$ is smaller than or equal to any other $J(\mathbf{x})$ inside the whole feasible region, $\mathbf{x}^*$ is called a *global optimum*.

When the function $J(\mathbf{x})$ is (at least) differentiable, minimas of functions are usually found by differencing the function and obtaining its gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$. Finding a location where the gradient equals zero is then the goal.

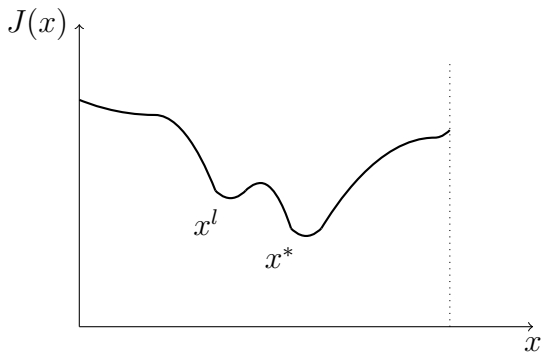

Figure 2.5: An example of a non-convex objective function with multiple local minimas. x^l and x^* are both local minimas. x^* is additionally the global minimum.

2.4.2 Gradient Descent methods

The basic *gradient descent* (GD), or *steepest descent*, optimization method is an iterative optimization method [14]. At each iteration, the steepest ascent direction (the gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$) is calculated. After that, a predefined step length α in the exactly opposite direction to the gradient is taken to decrease the value of the function. The update is described in Equation (2.6)

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} - \alpha_{(i)}\mathbf{d}_{(i)} = \mathbf{x}_{(i)} - \alpha_{(i)}\nabla_{\mathbf{x}} J(\mathbf{x}), \tag{2.6}$$

where $\mathbf{x}_{(i)} \in \mathbb{R}^{n\times 1}$ is the vector containing the variable values at iteration i, $\mathbf{d}_{(i)} \in \mathbb{R}^{n\times 1}$ is the descent direction at iteration i (the gradient direction) and $\alpha_{(i)}$ is a positive scalar. The step length $\alpha_{(i)}$ can be fixed or varied. A small step length gives an accurate solution, but requires more iterations, while a large step size might be too coarse and might even diverge. Varying the step size appropriately significantly decreases the required number of iterations. *Line-search methods* [14] are ways to evaluate α that should be used in Equation (2.6). The *Armijo* and *Wolfe conditions* for line-search [14] are common. The conditions state that the step length should give a sufficient decrease and curvature to be accepted for line search. If the conditions are not met, the step length is increased. The gradient descent method is well-known for often ending up zig-zagging towards the direction of the steepest descent, taking a long time to converge [14]. The gradient descent method continuously moves in the direction of the calculated negative gradient, stopping once no more improvement can be made in that direction. At these locations the new descent direction becomes orthogonal to the previous direction, which can result in the zig-zagging behaviour [14, 41]. There are numerous modifications to the standard gradient method. To name a few, Barzilai [10] considered a strategy of using only the previous step information to calculate the new direction. Preconditioning [58] can change the optimization problem to an easier problem.

2.4.3 The Newton methods

Newton's (Newton-Raphson) method [14] replaces step size α with the inverse of the Hessian matrix $\mathbf{H}^{-1}(\mathbf{x}) = \nabla^2_{\mathbf{x}} J(\mathbf{x})$, analogous to the inverse of the second derivative in one-dimensional space.

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} - \mathbf{H}^{-1}(\mathbf{x}_{(i)})\nabla_{\mathbf{x}} J(\mathbf{x}). \tag{2.7}$$

The Equation (2.7) is called the pure form of Newton's method. Newton's method converges superlinearly near local minima. Far from local minima, the Hessian matrix can be negative definite, resulting in ascent directions rather than descent directions. The pure form then needs modifications to converge globally [14]. Newton's method requires few iterations, but has an overhead in memory consumption and computational cost due to the calculation of the inverse of the Hessian.

2.4.4 The method of Conjugated Gradients

The *conjugate gradient* method [14, 41] solves the zig-zagging problem by introducing *conjugate* (*A-orthogonal*) search directions $\mathbf{d}_{(i)}$. The search direction are orthogonal with regard to the positive definite matrix $\mathbf{A}$:

$$\mathbf{d}_{(i)}^T \mathbf{A} \mathbf{d}_{(j)} = 0$$

The search directions are linearly independent of each other. These $\mathbf{A}$-conjugate search directions can be generated with the Gram-Schmidt search procedure [14]. The linear conjugate gradient is efficient at solving equations of the type

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.8}$$

where $\mathbf{A}$ is a positive definite matrix. These kind of equations occur naturally in quadratic forms, where the function subject to the minimization has the form

$$J(\mathbf{x}) = 1/2\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + \mathbf{c}. \tag{2.9}$$

The derivative of this function is precisely $\mathbf{A}\mathbf{x} - \mathbf{b}$ and by setting it to zero we obtain Equation (2.8). The conjugate gradient descent is therefore said to converge in exactly n iterations for quadratic forms. The steps in the conjugate gradient method for quadratic forms is described in Algorithm 1.

Algorithm 1: Conjugate gradient method for quadratic forms

Input: Positive Semidefinite matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, vector $\mathbf{b} \in \mathbb{R}^{n \times 1}$ and initial solution vector $\mathbf{x}_{(0)} \in \mathbb{R}^{n \times 1}$

Output: Solution vector $\mathbf{x}_{(n)}$

1. Calculate initial descent direction $\mathbf{d}_{(0)} = \mathbf{g}_{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(0)}$.

for $i = 0 \rightarrow (n-1)$ **do**

2. Calculate step length $\alpha_{(i)} = \dfrac{\mathbf{g}_{(i)}^T \mathbf{g}_{(i)}}{\mathbf{d}_{(i)}^T \mathbf{A}\mathbf{d}_{(i)}}$.

3. Update solution $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)}$.

4. Update negative gradient $\mathbf{g}_{(i+1)} = \mathbf{g}_{(i)} - \alpha_{(i)}\mathbf{A}\mathbf{d}_{(i)}$.

5. Update $\beta_{(i+1)} = \dfrac{\mathbf{g}_{(i+1)}^T \mathbf{g}_{(i+1)}}{\mathbf{g}_{(i)}^T \mathbf{g}_{(i)}}$.

6. Update descent direction $\mathbf{d}_{(i+1)} = \mathbf{g}_{(i+1)} + \beta_{(i+1)}\mathbf{d}_{(i)}$.

end for,

here $\mathbf{g}_{(i)}$ is the negative gradient at iteration i.

Conjugate gradient methods speed up the convergence without requiring the storage of the Hessian matrix. It is commonly used as an iterative procedure, terminating in less than n steps, because an adequate accuracy level has been reached [14]. The conjugate gradient method is perhaps the most prominent iterative method that solves sparse systems of linear equations.

2.4.5 Nonlinear Conjugated Gradients

The nonlinear conjugate gradient method [14, 41] is a direct generalization of the conjugate gradient method for functions $J(\mathbf{x})$ of which the gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$ can be calculated. The convergence of the nonlinear conjugate gradient is related to how well the function $J(\mathbf{x})$ approximates the quadratic form. [58]. The nonlinear conjugate gradient method is described in Algorithm 2.

Algorithm 2: Nonlinear conjugate gradient method

Input: Function $J(\mathbf{x})$, its gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$ and initial solution vector $\mathbf{x}_{(0)} \in \mathbb{R}^{n \times 1}$

Output: Solution vector $\mathbf{x}_{(T)}$

1. Calculate initial descent direction $\mathbf{d}_{(0)} = \mathbf{g}_{(0)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(0)})$.

for $i = 0 \rightarrow T$ **do**

2. Find $\alpha^*_{(i)} \geq 0$ s.t. $J(\mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)})$ is minimized (line search).

3. Update solution $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \alpha^*_{(i)}\mathbf{d}_{(i)}$.

4. Update negative gradient $\mathbf{g}_{(i+1)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(i+1)})$.

5. Update $\beta_{(i+1)}$.

6. Update descent direction $\mathbf{d}_{(i+1)} = \mathbf{g}_{(i+1)} + \beta_{(i+1)}\mathbf{d}_{(i)}$.

end for,

here $\mathbf{g}_{(i)}$ is the negative gradient at iteration i.

There are some common update methods for the parameter β: the Fletcher-Reeves [41], the Polak-Ribière method [41] and the Hestenes-Stiefel method [14]:

$$\beta^{FR}_{(i+1)} = \frac{\mathbf{g}^T_{(i+1)}\mathbf{g}_{(i+1)}}{\mathbf{g}^T_{(i)}\mathbf{g}_{(i)}}, \qquad \beta^{PR}_{(i+1)} = \frac{\mathbf{g}^T_{(i+1)}(\mathbf{g}_{(i+1)} - \mathbf{g}_{(i)})}{\mathbf{g}^T_{(i)}\mathbf{g}_{(i)}}, \qquad \beta^{HS}_{(i+1)} = -\frac{\mathbf{g}^T_{(i+1)}(\mathbf{g}_{(i+1)} - \mathbf{g}_{(i)})}{\mathbf{d}^T_{(i)}(\mathbf{g}_{(i+1)} - \mathbf{g}_{(i)})},$$

The Polak-Ribière method converges much faster than the Fletcher-Reeves method, however it can end up cycling infinitely without converging. Convergence is guaranteed in the Polak-Ribière method, if $\beta^{PR}_{(i+1)}$ is selected such that the $\beta = \max\{\beta^{PR}, 0\}$. When $\beta < 0$, the method is restarted, i.e. the gradient is calculated anew [58]. Because of non-quadratic terms in the cost function, the search directions progressively lose conjugacy and the search needs to be restarted periodically. The original update form, i.e. the Hestenes-Stiefel update, is still sometimes used. The conjugate gradient method has been found to have similarities with the Quasi-Newton BFGS method [47], discussed next.

2.4.6 Quasi-Newton methods: BFGS

Quasi-Newton methods [41] are used when the computation of the inverse Hessian matrix is impossible or is computationally too expensive. The Quasi-Newton methods have the update formula in Equation (2.10).

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} - \alpha_{(i)}\mathbf{S}_{(i)}(\mathbf{x}_{(i)})\nabla_{\mathbf{x}}J(\mathbf{x}) \tag{2.10}$$

The way that $\mathbf{S}_{(i)}$ is chosen makes the difference. If it is the identity matrix $\mathbf{S}_{(i)} = \mathbf{I}$, the method is called the gradient descent method. If it is the inverse Hessian, $\mathbf{S}_{(i)} = \mathbf{H}^{-1}_{(i)}$ the method is called Newton's method. If the Hessian of the objective function is not positive definite, the Hessian can be modified into a positive definite form and the resulting matrix can be used. This is sometimes called the modified Newton method. The efficiency depends on how accurate the estimate of the Hessian is. In neighbor embedding algorithms, such as t-SNE, a special form of the modified Newton method is efficiently used. The method is called the *Spectral directions* optimization [63], because it uses the positive definite matrix that is used in Laplacian Eigenmaps dimension reduction. Properties relating to eigenvalues are often referred to as spectral properties. The spectral direction optimization is efficiently coupled with Wolfe line-search methods [63].

The *Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm* [41] is one of the most famous nonlinear optimization methods. The update rules are stated in Algorithm 3. The method requires an initial guess $\mathbf{x}_{(0)}$. The initial Hessian estimate $\mathbf{S}_{(0)}$ can be any positive definite matrix.

Algorithm 3: BFGS algorithm for nonlinear optimization

Input: Function $J(\mathbf{x})$, its gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$ and an initial solution vector $\mathbf{x}_{(0)} \in \mathbb{R}^{n \times 1}$

Output: Solution vector $\mathbf{x}_{(T)}$

1. Assign initial inverse Hessian estimate $\mathbf{S}_{(0)} = \mathbf{I}$

2. Calculate initial negative gradient $\mathbf{g}_{(0)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(0)})$.

for $i = 0 \rightarrow T$ **do**

3. Calculate descent direction $\mathbf{d}_{(i)} = \mathbf{S}_{(i)}\mathbf{g}_{(i)}$.

4. Find $\alpha^*_{(i)} \geq 0$ s.t. $J(\mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)})$ is minimized (line search).

5. Calculate change in solution $\mathbf{p}_{(i)} = \alpha^*_{(i)}\mathbf{d}_{(i)}$.

6. Update solution $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{p}_{(i)}$.

7. Update negative gradient $\mathbf{g}_{(i+1)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(i+1)})$.

8. Calculate change in gradient $\mathbf{q}_{(i)} = -(\mathbf{g}_{(i+1)} - \mathbf{g}_{(i)})$.

9. Update inverse Hessian estimate

$$\mathbf{S}^{BFGS}_{(i+1)} = \mathbf{S}_{(i)} + \Big(\frac{1 + \mathbf{q}^T_{(i)}\mathbf{S}_{(i)}\mathbf{q}_{(i)}}{\mathbf{q}^T_{(i)}\mathbf{q}_{(i)}}\Big)\frac{\mathbf{p}_{(i)}\mathbf{p}^T_{(i)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}} - \Big(\frac{\mathbf{p}_{(i)}\mathbf{q}^T_{(i)}\mathbf{S}_{(i)} + \mathbf{S}_{(i)}\mathbf{q}_{(i)}\mathbf{q}^T_{(i)}}{\mathbf{q}^T_{(i)}\mathbf{p}_{(i)}}\Big).$$

end for

The Hessian estimate that the BFGS method produces is of rank two and it is symmetric. The search directions $\mathbf{d}_{(i)}$ can always be guaranteed to be descent directions by arranging that $\mathbf{S}_{(i)}$ is positive definite throughout the search process [41].

2.4.7 Memoryless Quasi-Newton methods: LBFGS

The *Limited-memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm* [41] is a fast algorithm, especially suited for optimization with a high number of variables. The LBFGS search procedure is presented in Algorithm 4. The algorithm does not store previous values of the inverse Hessian estimates, but sets them to the identity matrix $\mathbf{S}_{(i)} = \mathbf{I}$ in step 9, and is therefore called memoryless. The method is reset every n iterations. The steps 3 and 9 can be combined, yielding the descent direction update rule in Equation (2.11)

$$\mathbf{d}_{(i+1)} = -\mathbf{g}_{(i+1)} + \frac{\mathbf{q}_{(i)}\mathbf{p}^T_{(i)}\mathbf{g}_{(i+1)} + \mathbf{p}_{(i)}\mathbf{q}^T_{(i)}\mathbf{g}_{(i+1)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}} - \Big(1 + \frac{\mathbf{q}^T_{(i)}\mathbf{q}_{(i)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}}\Big)\frac{\mathbf{p}_{(i)}\mathbf{p}^T_{(i)}\mathbf{g}_{(i+1)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}} \qquad (2.11)$$

Algorithm 4: LBFGS algorithm for nonlinear optimization

Input: Function $J(\mathbf{x})$, its gradient $\nabla_{\mathbf{x}} J(\mathbf{x})$ and an initial solution vector $\mathbf{x}_{(0)} \in \mathbb{R}^{n \times 1}$

Output: Solution vector $\mathbf{x}_{(T)}$

1. Assign initial inverse Hessian estimate $\mathbf{S}_{(0)} = \mathbf{I}$

2. Calculate initial negative gradient $\mathbf{g}_{(0)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(0)})$.

for $i = 0 \rightarrow T$ **do**

3. Calculate descent direction $\mathbf{d}_{(i)} = \mathbf{S}_{(i)}\mathbf{g}_{(i)}$.

4. Find $\alpha^*_{(i)} \geq 0$ s.t. $J(\mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)})$ is minimized (line search).

5. Calculate change in solution $\mathbf{p}_{(i)} = \alpha^*_{(i)}\mathbf{d}_{(i)}$.

6. Update solution $\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \mathbf{p}_{(i)}$.

7. Update negative gradient $\mathbf{g}_{(i+1)} = -\nabla_{\mathbf{x}} J(\mathbf{x}_{(i+1)})$.

8. Calculate change in gradient $\mathbf{q}_{(i)} = -(\mathbf{g}_{(i+1)} - \mathbf{g}_{(i)})$.

9. Update inverse Hessian estimate

$$\mathbf{S}^{LBFGS}_{(i+1)} = \mathbf{I} - \frac{\mathbf{q}_{(i)}\mathbf{p}^T_{(i)} + \mathbf{p}_{(i)}\mathbf{q}^T_{(i)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}} + \Big(1 + \frac{\mathbf{q}^T_{(i)}\mathbf{q}_{(i)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}}\Big)\frac{\mathbf{p}_{(i)}\mathbf{p}^T_{(i)}}{\mathbf{p}^T_{(i)}\mathbf{q}_{(i)}}.$$

end for

If the line search in step 4 is exact, the update rule in Equation (2.11) reduces to a very similar form as in the Polak-Ribière conjugate gradient search. Experimentally, the inexact line search has been shown to be more efficient [41]. In practice, nonlinear CG and LBFGS are two competing algorithms in nonlinear optimization with a large number of variables. LFBGS usually requires fewer iterations than CG, and is especially useful when the function evaluations are expensive to calculate.

2.4.8 Linear and semidefinite programming

The previous optimization techniques were targeted to unconstrained optimization problems. Linear programming solves mathematical problems with linear constraints (linear programs) [41]. There are many ways of formulating a linear program. One way to formulate it is by using inequations [62]:

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\text{s.t } \mathbf{A}\mathbf{x} + \mathbf{b} \geq 0 \end{aligned} \tag{2.12}$$

here the inequation is a component-wise inequation. Linear programs occur widely in many fields. For example, finding a separating hyperplane in support vector machines, as well as manufacturing and warehouse optimization problems are linear programs [41]. The simplex and interior-point methods are famous solvers for linear programs.

Semidefinite programs contain linear objective function and can be written in the form [62]:

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\text{s.t } F(\mathbf{x}) \geq 0 \end{aligned} \tag{2.13}$$

here $F(\mathbf{x}) \geq 0$ is a semidefinite inequation. Linear programs are special cases of semidefinite programs, where the semidefinite inequation $F(\mathbf{x}) \geq 0$ is a linear inequation $\mathbf{Ax}+\mathbf{b} = 0$. Many nonlinear optimization problems can be stated as semidefinite programming problems. Semidefinite programs can be solved very efficiently, both theoretically and in practice [62]. Interior-point methods are commonly used in solving semidefinite programs.

2.5 Information Theoretical Concepts

This section elaborates key information theoretical concepts [18] that will be used in the analysis to follow. Namely, the information entropy and the Kullback-Leibler divergence are discussed with example.

2.5.1 Entropy

Information theoretical entropy is a measure of how much uncertainty a probability distribution contains. The more similar a probability distribution $p(x)$ is to the uniform distribution $u(x)$, the higher the entropy for discrete variables. This is because the uniform distribution $u(x)$ contains the least prior information about which state a variable x is in. [8] The entropy can be calculated for discrete variables x as

$$\mathbb{H}(x) = -\sum_{i=1}^{N} p(x) log(p(x)) \tag{2.14}$$

The entropy is a measure of average uncertainty in a random variable. If the logarithm used is 2, the entropy is the average number of bits required to describe a random variable. If it is described using the natural logarithm, it is the average amount of nats required to describe the random variable. Nats can be changed to bits by multiplication with an appropriate factor and vice versa. The following example is a modified version supplied in the book "Elements of Information Theory" [18], which illustrates the concept of entropy particularly well.

Example (1). There are 8 banks in a large city, where break-ins occur with probabilities $[\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}]$. A distress signal used to identify the bank would have the length 3 bits, as there are $2^3 = 8$ different alternatives. However, given the fact of the break-in probability distribution we can write it to be more efficient *in average*. One such encoding could be $1, 01, 001, 000100, 000110, 000101, 000111$, which has the average length 2 bits. The entropy of the probability distribution is:

$$\mathbb{H}(X) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{8} - 4\frac{1}{16}\log\frac{1}{16} = 2 \text{ bits.} \tag{2.15}$$

which is 2 bits as well. We can say that the information is more structured than it would be if the break-ins were to occur with uniform probability.

For continuous variables, the Gaussian distribution has the largest entropy among all distributions that have the same covariance matrix. In this sense, it is the most random distribution containing the least amount of structured information. Entropy for

continuous variables is defined as in Equation (2.14), but the sum is replaced by an integral over x.

2.5.2 Cross-entropy

The cross-entropy [45] $\mathbb{H}(p,q) = -\sum_{i=1}^{n} p(x)log(q(x))$ is the average number of bits needed to encode a signal coming from a distribution $p(x)$, when using another codebook $q(x)$ to encode it. Continuing the previous sample, the cross-entropy of encoding the bank distress signals with a uniform distribution $u(x)$ would have been

$$\mathbb{H}(p,u) = -\frac{1}{2}\log\frac{1}{8} - \frac{1}{4}\log\frac{1}{8} - \frac{1}{8}\log\frac{1}{8} - \frac{4}{16}\log\frac{1}{8} = -1\log\frac{1}{8} = 3 \text{ bits.} \tag{2.16}$$

2.5.3 Kullback-Leibler divergence

Information divergence is a way to measure the distance between two probability distributions. The divergence can be calculated for both discrete distributions and for continuous distributions. The most commonly used information divergence is the Kullback-Leibler divergence (the relative entropy). Divergences are in general not symmetric, i.e. the error in approximating a distribution Q with another distribution P is often different than vice versa [8].

The relative entropy is always non-negative and zero only in the case that $Q = P$. However, the relative entropy is not a real distance because it does not fulfill the triangle inequality and it is not symmetric. The Kullback-Leibler divergence for two discrete probability mass functions P and Q is

$$D_{KL}(P||Q) = \sum_{i}^{n} p_i \log(p_i/q_i). \tag{2.17}$$

The Kullback-Leibler divergence can be written in terms of the entropy $\mathbb{H}(P)$ and the cross-entropy $\mathbb{H}(P,Q)$: $D_{KL}(P||Q) = \mathbb{H}(P,Q) - \mathbb{H}(P)$. It is just the average number of extra bits needed to encode a signal from a probability distribution P with a codebook using probability distribution Q. In the previous example, the Kullback-Leibler divergence would have been 1. The Kullback-Leibler divergence is often used as an optimization criterion to make a model distribution Q similar to a target distribution P. This KL divergence $D_{KL}(P||Q)$ is sometimes called M-projection (moment projection) [45]. The M-projection is infinite if q_i is zero, therefore situations where q_i would be very small are avoided. The M-projection is called zero-avoiding [45].

If the probability distribution P is difficult to evaluate, the reverse Kullback-Leibler divergence $D_{KL}(Q||P) = \sum_{i}^{n} q_i \log(q_i/p_i)$ can be used to measure the distance between the distributions. The KL divergence is not symmetric in its arguments and reversing the order of the arguments causes a different cost. The reverse KL divergence $\sum_{i=1}^{n}\sum_{i=1}^{n} q_i \log(q_i/p_i)$ is called I-projection (information projection). The I-projection is infinite if p_i is zero, which means that small values in p_i need to be compensated with small values in q_i. The I-projection is called zero-forcing [45].

Chapter 3

Supervised Distance Preserving Projections

This chapter introduces the dimension reduction method called Supervised Distance Preserving Projections [66]. While SDPP is primarily a dimension reduction method for regression datasets, it can also be formulated for classification problems. Here, the focus is put on the more traditional classification setting. The discussion serves as an introduction to the proposed method. In Section 4.1, some of these problems are solved by changing the cost function to match probabilities rather than squared Euclidean distances.

3.1 Introduction

The SDPP is a supervised dimensionality reduction technique, especially designed for regression tasks. It is motivated by the Weierstrass definition of continuity for functions. The definition states that if two points are close in the co-variate space then they should be close in the response space too. The assumption used is that the data points are sampled frequently enough that such a continuity can be observed. The Weierstrass definition for continuity is often called the (ϵ, δ)-definition for continuity. Formally, given a continuous mapping of $X \to f$, there exist positive real numbers ϵ and δ, such that $|\mathbf{x} - \mathbf{x}'| < \epsilon \implies |f(\mathbf{x}) - f(\mathbf{x}')| < \delta$, where $\mathbf{x}, \mathbf{x}'$ are close points. Both ϵ and δ should be relatively small numbers. In SDPP a linear subspace that preserves this relationship is pursued.

3.2 Cost Function

The SDPP optimizes a transformation matrix $\mathbf{W} \in \mathbb{R}^{D \times D}$, given an input matrix $\mathbf{X} \in \mathbb{R}^{D \times n}$ and its response matrix $\mathbf{Y} \in \mathbb{R}^{d_y \times n}$. Here, n is the number of observations, D is the number of variables and d_y is the number of response variables. Typically $n \gg D > d_y$, because we assume that the data is sampled frequent enough. The optimization is done so that the distances between data points in the subspace $\mathbf{Z} = \mathbf{W}^T\mathbf{X} \in \mathbb{R}^{D \times n}$ mimic the distances between data points in the response space $\mathbf{Y}$. The response matrix is of the form $\mathbf{Y} = [\mathbf{y}_1\mathbf{y}_2...\mathbf{y}_n] \in \mathbb{R}^{d_y \times n}$ if the problem is a regression problem and of the form

$\mathbf{Y} = [\mathbf{y}_1\mathbf{y}_2...\mathbf{y}_n] \in \mathbb{I}^{d_y \times n}$ if the problem is a classification problem, where $\mathbb{I}^{d_y \times n}$ is the set of indicator variables. The optimization criterion of SDPP is the following:

$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i)} (d_{ij}^2(\mathbf{W}) - \delta_{ij}^2)^2 \tag{3.1}$$

where the projected input space distances are $d_{ij}(\mathbf{W}) = ||\mathbf{z}_i - \mathbf{z}_j||_2 = ||\mathbf{W}^T\mathbf{x}_i - \mathbf{W}^T\mathbf{x}_j||_2$, the response space distances are $\delta_{ij} = ||\mathbf{y}_i - \mathbf{y}_j||_2$ and $\mathcal{N}(\mathbf{x}_i)$ is a set of closely lying points to $\mathbf{x}_i$. In practice, the close points are evaluated by performing a nearest-neighbor search in the input space. The squared Euclidean distance is measuring how far apart the squared distances between data points on the $\mathbf{Z}$-side and $\mathbf{Y}$-side are from each other. The optimization criterion can be written with a neighbor graph as

$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{G}_{ij}(\mathbf{D}_{ij}^2(\mathbf{W}) - \mathbf{\Delta}_{ij}^2)^2 \tag{3.2}$$

where $\mathbf{G} \in \mathbb{R}^{n \times n}$ is matrix containing n^2 indicator variables telling if a data point j is a neighbor of data point i, $\mathbf{D} \in \mathbb{R}^{n \times n}$ are the distances in the projected input space and $\mathbf{\Delta} \in \mathbb{R}^{n \times n}$ are the distances in the response space. The cost function in SDPP is similar to the one in linear regression with the exception that there are n^2 variables that are fitted to each other. SDPP also has a kernel extension [66].

3.2.1 SDPP for classification data sets

In classification data sets, the response matrix $\mathbf{Y}$ contains categorical information. A common approach is to gather all data points of the same class close and to keep other data points further away by assigning the *ideal distances* as in Equation (3.3).

$$\delta_{ij} = \begin{cases} 0, & \text{if } \mathbf{y}_i = \mathbf{y}_j \\ 1, & \text{if } \mathbf{y}_i \neq \mathbf{y}_j \end{cases} \tag{3.3}$$

where the response variables $\mathbf{y}_i$ are sequences of d_y binary numbers, specifying the class labels. This way of encoding distances creates regular structure, a simplex. Ideally, all data points of the same cluster are placed on top of each other (zero distance) and the distance between these points to all other points of different classes is set to 1. A simplex created by imposing these conditions is shown in Figure 3.1. The length of each edge is exactly 1. Figure (3.2) shows three classes embedded with SDPP using this principle.

3.2.2 The neighborhood matrix G

A neighborhood matrix appears in the cost function of SDPP. The neighborhood matrix restricts which pairwise distances are allowed to enter the cost and which are left outside. The choice of the neighborhood matrix affects the resulting projection matrix. There are different ways of calculating the neighborhood matrices. The most common type is of the k-nearest neighborhood matrix. For each data point $\mathbf{x}_i$, the k nearest data points $\mathbf{x}_j$ are calculated. The distances are usually calculated as Euclidean distances (L_2-distance). The index values I_i that belong to the k-nearest neighbors of $\mathbf{x}_i$ get values $\mathbf{G}_{ij} = 1, \forall j \in I_i$ and all other elements get the value zero. The k-$\mathcal{NN}$ graph is not symmetric. It can

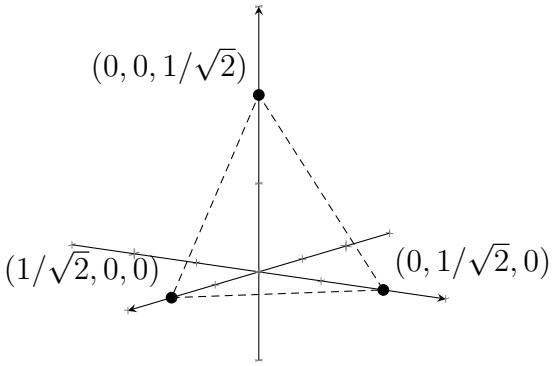

Figure 3.1: Imposing equidistance between clusters of data creates a simplex structure. Each point represents a superposition of all data points of the same class. The distance between all clusters is ideally the same.

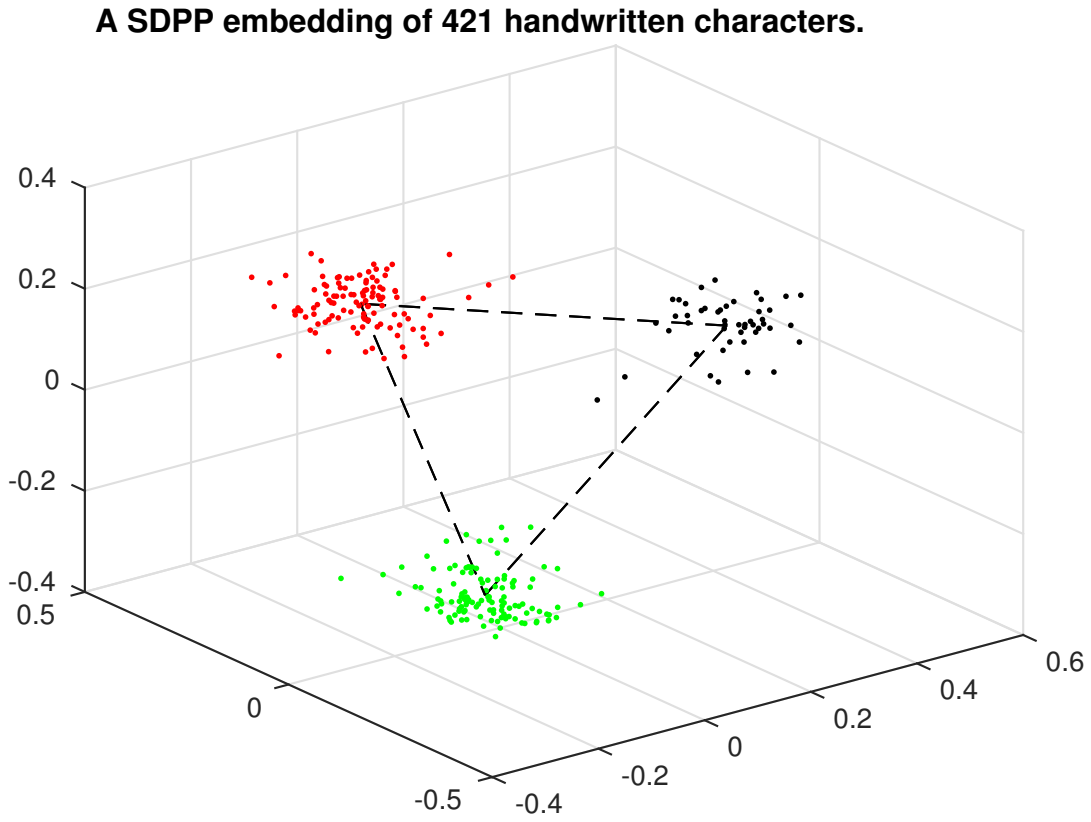

Figure 3.2: A SDPP embedding of 3 classes in a 3-dimensional space.

be made symmetric by calculating the symmetric nearest neighbors $\hat{\mathbf{G}}_{ij} = (\mathbf{G}_{ij} + \mathbf{G}_{ij}^T)/2$ or the mutual nearest neighbors: $\bar{\mathbf{G}}_{ij} = \min(\mathbf{G}_{ij}, \mathbf{G}_{ij}^T)$. A k mutual nearest neighbor connection $\bar{\mathbf{G}}_{ij}$ needs to be within the k nearest neighbors of both $\mathbf{x}_i \to \mathbf{x}_j$ and $\mathbf{x}_j \to \mathbf{x}_i$.

There are other ways of calculating neighbors, too. The *natural neighbors* ($\mathcal{NN}$) and *enclosing k-nearest neighbors* (ek$\mathcal{NN}$) are generally difficult to compute as they scale badly with higher-dimensional data. Both methods require the calculation of the convex hull in the high-dimensional space [23, 24]. The time complexity of the calculation of the convex hull scales exponentially with both the number of samples and the number of dimensions, making it computationally too demanding in high-dimensional spaces. The geometry of a high-dimensional space makes the number of neighbors obtained with $\mathcal{NN}$ and ek$\mathcal{NN}$ to a data point increase dramatically with each added dimension, overestimating the size of the neighborhood in high-dimensional space.

3.3 Optimization

The SDPP can be optimized with two different strategies: by the nonlinear conjugate gradient descent method (CG) or by solving a semidefinite (mathematical) program (SDP/SQLP) [60, 62]. In practice, optimization by SQLP is very fast, but creating of the constraints is time consuming. The SQLP solver does not need an initial solution, which makes it preferable in some sense. In practice, the CG and SQLP solutions produce quite similar results as long as the initialization in CG does not contain zeros.

3.3.1 SQLP formulation

The SDPP can be solved by quadratic semi-definite programming [60, 62]. The squared distances in the transformed input space are written as $d_{ij}(\mathbf{W}) = ||\mathbf{W}^T\mathbf{x}_i - \mathbf{W}^T\mathbf{x}_j||_2 = (\mathbf{W}^T(\mathbf{x}_i - \mathbf{x}_j))^T\mathbf{W}^T(\mathbf{x}_i - \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)\mathbf{W}\mathbf{W}^T(\mathbf{x}_i - \mathbf{x}_j)$, where $\mathbf{P} = \mathbf{W}\mathbf{W}^T$ is positive semidefinite (psd). The distances can be stated as a vector inner product by vectorizing the matrix $\mathbf{l}_{ij} = \text{vec}(\boldsymbol{\tau}_{ij}\boldsymbol{\tau}_{ij}^T)$ and $\mathbf{p} = \text{vec}(\mathbf{P})$, with $\boldsymbol{\tau}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$: $d_{ij}(\mathbf{W}) = \mathbf{l}_{ij}^T\mathbf{p}$. Then the problem can be stated as a quadratic form in Equation (3.4):

$$J(\mathbf{p}) = \mathbf{p}^T\mathbf{A}\mathbf{p} + \mathbf{b}^T\mathbf{p} + c, \tag{3.4}$$

with $\mathbf{A} = (\sum_{ij}^n \mathbf{G}_{ij}\mathbf{l}_{ij}\mathbf{l}_{ij}^T)/n$, $\mathbf{b} = -2(\sum_{ij}^n \mathbf{G}_{ij}\delta_{ij}^2\mathbf{l}_{ij})/n$ and $c = (\sum_{ij}^n \mathbf{G}_{ij}\delta_{ij}^4)/n$. In the expression, c is a constant that can be ignored in the optimization. The matrix $\mathbf{A}$ is positive definite if $\mathbf{G}$ is symmetric. The number of variables optimized is D^2. Being a quadratic form, the optimization can be performed in exactly D^2 steps with the linear conjugate gradient algorithm or in one step with Newton's algorithm. But for Newton's algorithm to work the Hessian $H(\mathbf{p}) = \mathbf{A} \in R^{D^2 \times D^2}$ would need to be inverted, which is difficult if the input dimensionality is large. The problem is actually a semidefinite program because $\mathbf{P}$ is positive semidefinite. This makes it possible to transform the problem into a SDP optimization problem with a convex cone constraint, yielding a faster optimization routine (SQLP) [60]. The SQLP formulation of the optimization problem is presented in Equation (3.5):

$$\begin{aligned} \min_{\mathbf{p},\mathbf{u}} & (\mathbf{e}_1 - \mathbf{e}_2)^T\mathbf{u} + \mathbf{b}^T\mathbf{p} \\ \text{s.t. } & (\mathbf{e}_1 + \mathbf{e}_2)^T\mathbf{u} = 1, \\ & \mathbf{B}\mathbf{p} - \mathbf{C}\mathbf{u} = 0, \\ & \mathbf{u} \in K_{q+2}, \\ & \mathbf{P} \text{ is psd} \end{aligned} \tag{3.5}$$

Here q is the rank of matrix $\mathbf{A}$. The size of the matrix $\mathbf{B}$ is $D^2 \times D^2$, where D is the original dimensionality of the data matrix, that is D in $\mathbf{X} \in \mathbb{R}^{D\times n}$. In problems where the dimensionality is high ($D > 100$), the memory requirements of temporary storage of the matrix $\mathbf{A}$ become inhibitively large. The nonlinear conjugate gradient descent method is recommended in these occasions [66]. The optimization workflow for SQLP is described in Algorithm 5 [66].

Algorithm 5: SQLP optimization for SDPP

Input: Input matrix $\mathbf{X}$, response matrix $\mathbf{Y}$ and neighborhood matrix $\mathbf{G}$
Output: Projection matrix $\mathbf{W}$

1. Form vector $\boldsymbol{\tau}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$
2. Vectorize $\mathbf{l}_{ij} = \text{vec}(\boldsymbol{\tau}_{ij}\boldsymbol{\tau}_{ij}^T)$
3. Form matrix $\mathbf{A} = 1/n(\sum_{ij}^n \mathbf{G}_{ij}\mathbf{l}_{ij}\mathbf{l}_{ij}^T)$
4. Form vector $\mathbf{b} = -2/n(\sum_{ij}^n \mathbf{G}_{ij}\delta_{ij}^2\mathbf{l}_{ij})$
5. Form matrix $\mathbf{B}$ by Cholesky factorizing $\mathbf{A} = \mathbf{B}^T\mathbf{B}$.
6. Form matrix $\mathbf{C} = [\mathbf{0}_{q\times 2}, \mathbf{I}_{q\times q}]$.
7. Solve the SQLP problem stated in Equation (3.5).
8. Reshape vector $\mathbf{p}$ into matrix $\mathbf{P}$
9. Singular Value Decompose $\mathbf{P} = \mathbf{USV}$
10. Return projection matrix $\mathbf{W} = \mathbf{US}$.

3.3.2 Nonlinear Conjugate Gradient descent

The gradient of the cost function in Equation (3.2) is

$$\nabla_{\mathbf{W}} J = \frac{4}{n}\sum_{ij} \mathbf{G}_{ij}(\mathbf{D}_{ij} - \boldsymbol{\Delta}_{ij})\boldsymbol{\tau}_{ij}\boldsymbol{\tau}_{ij}^T\mathbf{W} \tag{3.6}$$

It can be written with the graph Laplacian in the form

$$\nabla_{\mathbf{W}} J = \frac{4}{n}\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W}, \tag{3.7}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{R}^+ \in \mathbb{R}^{n\times n}$ with

$$\begin{aligned} \mathbf{R} &= \mathbf{G} \odot (\mathbf{D} - \boldsymbol{\Delta}) \\ \mathbf{R}^+ &= \mathbf{R} + \mathbf{R}^T \\ \mathbf{D} &= \sum_i \mathbf{R}_{ij}^+ \end{aligned} \tag{3.8}$$

Here $\odot$ denotes the Hadamard product (element-wise multiplication) and $\mathbf{R}^+$ is a symmetrized matrix of $\mathbf{R} \in \mathbb{R}^{n \times n}$, and $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the row sum of $\mathbf{R}^+$. The solution $\mathbf{W}$ is iteratively optimized by the conjugate gradient method. Algorithm 6 is the updated version of the nonlinear conjugate gradient descent appearing in the paper *Supervised Distance Preserving Projections* [66] by Corona et al.

Algorithm 6: Conjugate gradient optimization for SDPP

Input: Input matrix $\mathbf{X}$ and response matrix $\mathbf{Y}$, neighborhood matrix $\mathbf{G}$ and initial projection matrix $\mathbf{W}_0$
Output: Projection matrix $\mathbf{W}$
1. Compute gradient $\nabla_{\mathbf{W}} J$ with Equation (3.7)
2. Vectorize projection matrix, $\mathbf{w}_0 = \text{vec}(\mathbf{W}_0)$
3. Vectorize gradient, $\mathbf{g}_0 = \text{vec}(\nabla_{\mathbf{W}} J)$
4. Initialize the conjugate direction as $\boldsymbol{\nu}_0 = -\mathbf{g}_0$
for $t = 1 \rightarrow T$ **do**
5. Calculate β_t by Polak-Ribière's rule, $\beta_t = \dfrac{\mathbf{g}_t^T(\mathbf{g}_t - \mathbf{g_{t-1}})}{\mathbf{g}_t^T \mathbf{g}_{t-1}}$
6. Update the conjugate direction, $\boldsymbol{\nu}_t = -\mathbf{g}_t + \beta_t \boldsymbol{\nu}_{t-1}$
7. Perform line search, $\eta_t = \text{argmin}_\eta J(\mathbf{w} + \eta \boldsymbol{\nu}_t)$
8. Update $\mathbf{w}_{t+1} = \mathbf{w} + \eta_t \boldsymbol{\nu}_t$
end for
9. Reshape the vector $\mathbf{w}_{T+1}$ into matrix $\mathbf{W}$
10. Decompose with thin SVD $\mathbf{P} = \hat{\mathbf{U}}\hat{\mathbf{S}}\mathbf{V}$
11. Return projection matrix $\mathbf{W} = \hat{\mathbf{U}}\hat{\mathbf{S}}$.

3.4 Evaluation of classification solutions with SDPP

In SDPP the objective function tries to match *squared* distances in the input space with *squared* distances in the output space. The squared distances give a very steeply rising cost function. The cost function is similar to a *barrier* function used in constrained optimization. Interpreted in this way, the cost function defines that the data points of the same class *must be* lying on top of each other and the data points of different classes *must be* on a predefined distance from each other (for those points that the optimization problem sees in the neighbor graph).

The Iris dataset [5] contains three Iris subspecies in four dimensions. Each subspecies constitutes a class. There are 50 data points in each class. The dataset is sorted, so that class 1 occupies indices 1 to 50, class 2 occupies indices 51 to 100 and class 3 the indices 101 to 150. The Iris dataset is a simple dataset that is often used to demonstrate the suitability of dimension reduction methods. One of the flower species is linearly separable from the other two, while two of the flower species are partially occupying the same areas in the space.

Figure 3.3 shows a SDPP embedding using the Iris dataset. The neighborhood graph is calculated using 15 nearest neighbors of each data point. Looking at Figure 3.3 it can be seen that the cost function only sees some of all the within-class distances for each class and also some between-class distances between class 2 and 3. Black points in the neighborhood matrix signal that the distances between the points enter the cost function.

The graph is not symmetric. The neighborhood connections shows symmetric within-class links as dark blue and one-sided links as light blue. The between class links are red. The red class has no connections to other classes, which means that the distances between green and red or blue and red are not part of the optimization problem.

The neighbor graph defines the geometry that is pursued in the optimization problem. The optimal geometry can be seen in Figure 3.4. Simply put, classes 2 and 3 (blue and green) should be separated, while class 1 (red) is unconstrained. Obviously, because class 3 is unconstrained, it could end up lying on top of classes 2 or 3 in the worst case. Having a small amount of neighbors is risky in high-dimensional classification settings. It turns out that the classes are fairly well separated in the original space and there is no problem with overlapping classes.

Figure 3.5 shows another embedding provided by SDPP on the Iris dataset. The neighborhood graph is calculated with the 50 mutual nearest neighbors and is symmetric. The green and blue classes are heavily connected in the neighborhood matrix and a few data points in the red class are connected to only one data point in the green class. Because of the steeply rising penalty in the cost function, the absolute amount of connections between the different classes are less important. Rather, the question is which classes have connections as defined in the neighbor graph. The scale of the coordinate axes also changes as a result of the optimal geometry. Scaled equally, the projection would actually be a *line*: all data points are in the range $[-1, 3]$ in the first dimension and in the range $[0.04, 0.14]$ in the second dimension. The simplified geometry is seen in Figure 3.6. The geometry defined with this kind of neighbor graph makes it possible to embed the Iris dataset into one dimension (a line).

Let us inspect what a geometry that accepts connections from all three classes looks like. The neighbor graph is the modified graph in embedding 2, with an added connection between the closest data points in class 1 and class 2 (required to be mutual). The resulting embedding is shown in Figure 3.7 and the optimal structure is shown in Figure 3.8. Adding the extra connections pulls the third class closer to the first class, making the third class encroach upon the area of the second class, blurring the class boundary between classes 2 and 3. Unfortunately, the optimal structure is not achieved in this embedding.

3.5 Personal work related to the SDPP

During the summer 2014, I had the opportunity to work as a summer student in the Information and Computer Science department at Aalto University. This included other work that is not included in this thesis. Some of the work is summarized here. The existing SDPP code was reformatted into a toolbox, available at the ICS webpage [40]. The calculation of the sequential quadratic program version of SDPP was speeded up by rewriting calculations of the constraints more efficiently. The constraints were calculated only for the elements with non-zero neighbor connections. The calculations were scaled to different sized machines by analyzing the elements in the large matrices and automatically splitting the additive elements into separate operations (allowing parallelisation). A considerable speedup was achieved without the need to approximate the solution. The nonlinear conjugate gradient method was speeded up with the same idea. The squared distances calculation was speeded up by only calculating the parts constituting the non-zero elements in the neighborhood graph. This part constitutes the most expensive part in

the iterative method. The cost function of SDPP was also updated to accurately account for non-binary neighborhood values.

The SDPP toolbox includes many different ways of calculating the neighborhood matrix: the k-nearest neighbors, the k-nearest mutual neighbors, natural neighbors and an experimental nearest correlation neighborhood. In the last method, the idea is to find neighboring points that approximate a straight line in the data [22].

I worked a couple of weeks on an online version of SDPP. The method used stochastic gradient descent for optimization. Each point-to-point distance was added to the cost function sequantially after which the optimization was done for this distance. The sequence approached the right range of the optimal solution, but it did not converge to the minimum. A mini-batch version of the problem did not converge either. Stochastic gradient descent has asymptotic convergence on convex problems. The reason that it did not converge is likely due to the fact that SDPP has two global optimal solutions: $\mathbf{W}^*$ and $-\mathbf{W}^*$, making it non-convex when $\mathbf{W}^* \neq \mathbf{0}$. The optimization process was very slow compared to the conjugate gradient version and the work was discontinued.

The SDPP was extended to work with another type of distance, the Fisher-Rao distance [17] between two probability distributions. The distributions were calculated with the perplexity search described by van der Maaten [61]. Fisher-Rao distances between data points then were a function of both the distance between points and the density of the point cloud region of the two points. The distance was directly applied to the SDPP function. The embedding results were more accurate than plain SDPP solutions in regression test problems, but the calculation of the embedding was slower. The SDPP with Fisher-Rao distance worked poorly on classification problem and work on it was discontinued.

There was work conducted on including a Gaussian distributed neighborhood matrix, such as the one included in SNE. The neighbor graph then had all positive values, with the value shrinking exponentially fast the farther away distances were. This clashed with the curse of dimensionality [19], namely in that the distances between points became increasingly similar in high-dimensional space, see Figure 3.9. The neighborhoods are much larger than the perplexity value would indicate and incurred very similar projections in SDPP. The calculation with different values of perplexity became dubious in SDPP, as the perplexity value did not correspond to the intuitive idea of the *efficient* number of neighbors, presented in [61].

3.6 Conclusions on SDPP

The SDPP is an attractive supervised dimensionality reduction technique for moderately high-dimensional problems. SDPP can be used for both regression and classification problems. SDPP relies heavily on the geometry described in the neighbor graph. The neighbor graph is affected by the curse of dimensionality, since it is calculated in the original high-dimensional space and is not updated. The sense of proximity becomes increasingly likely to be affected by the noisy variables, making the use of neighbor graphs risky. In SDPP the fact that certain distances enter the cost can change the visualization to a great extent (compare figures 3.5 and 3.7). Stochastic Discriminant Analysis discussed in the next chapter tries to avoid problems faced when using multiclass data in high-dimensional settings by matching probabilities instead of distances.

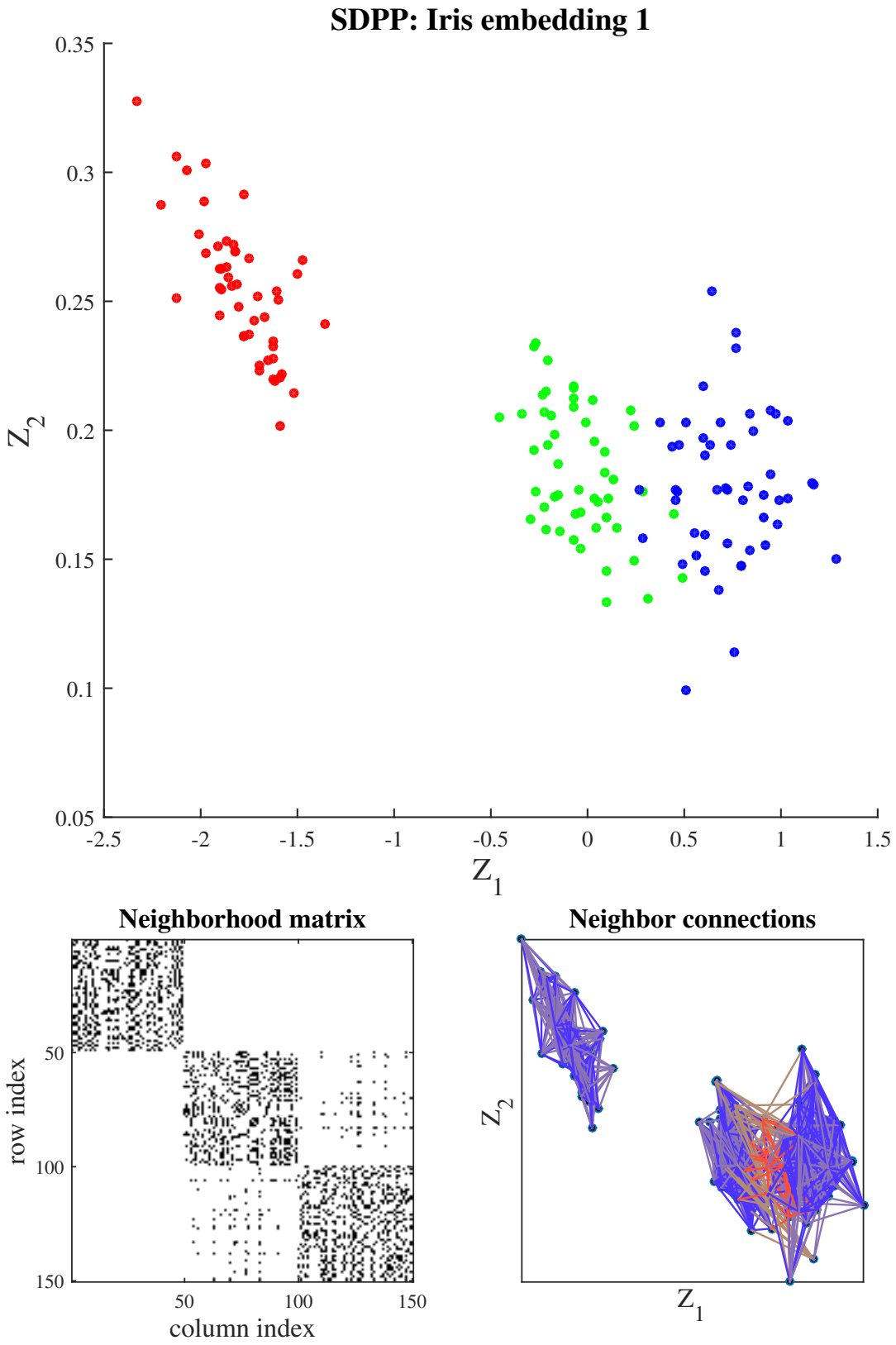

Figure 3.3: A projection of the Iris data set onto 2D.

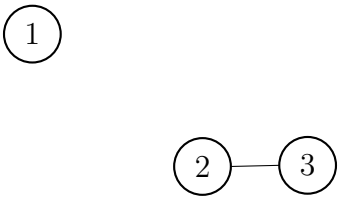

Figure 3.4: Optimal structure as seen in Iris embedding 1.

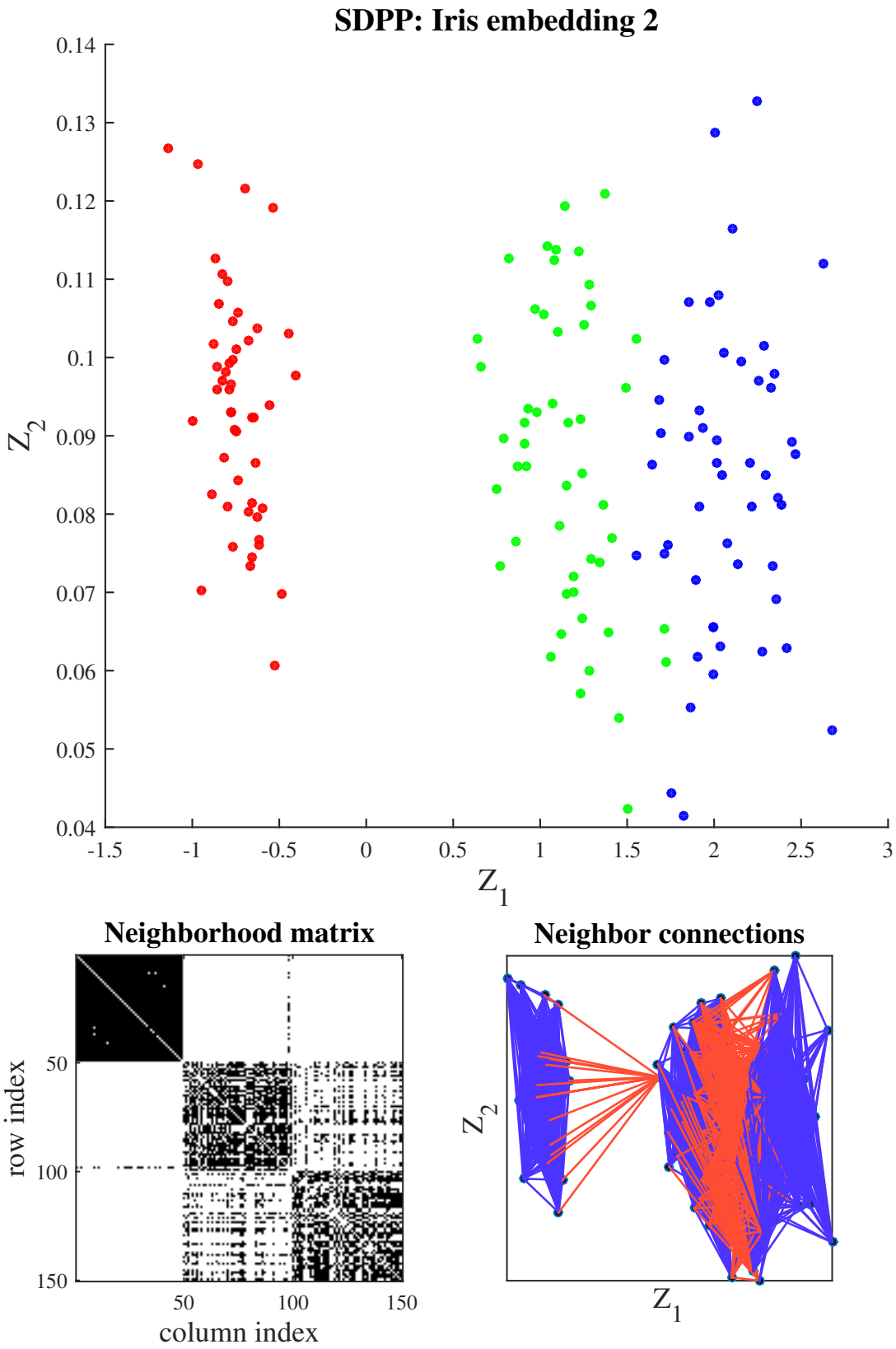

Figure 3.5: Another projection of the Iris data set onto 2D.

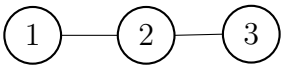

Figure 3.6: Optimal structure as seen in Iris embedding 2.

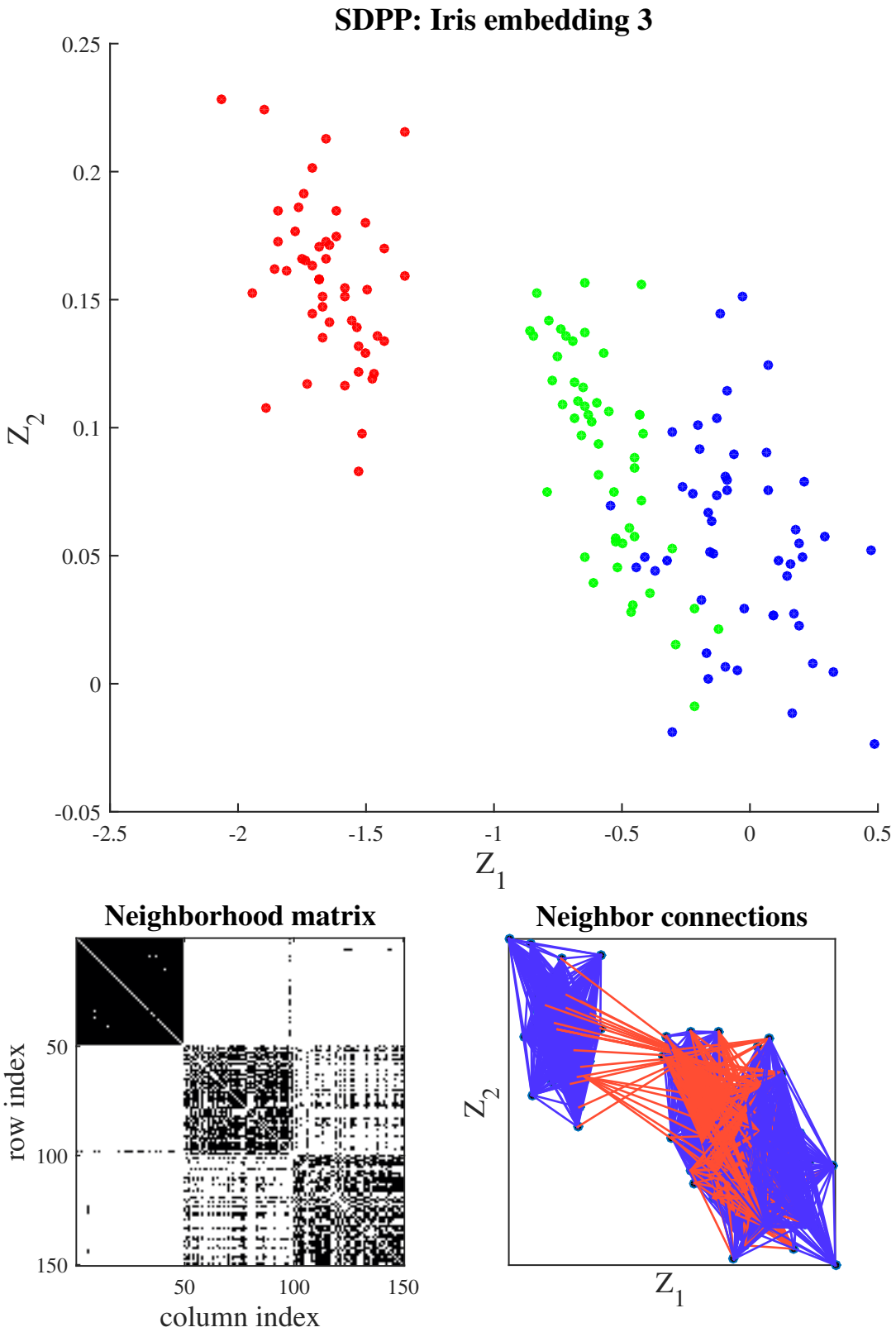

Figure 3.7: Yet another projection of the Iris data set onto 2D.

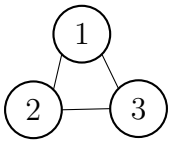

Figure 3.8: Optimal structure as seen in Iris embedding 3.

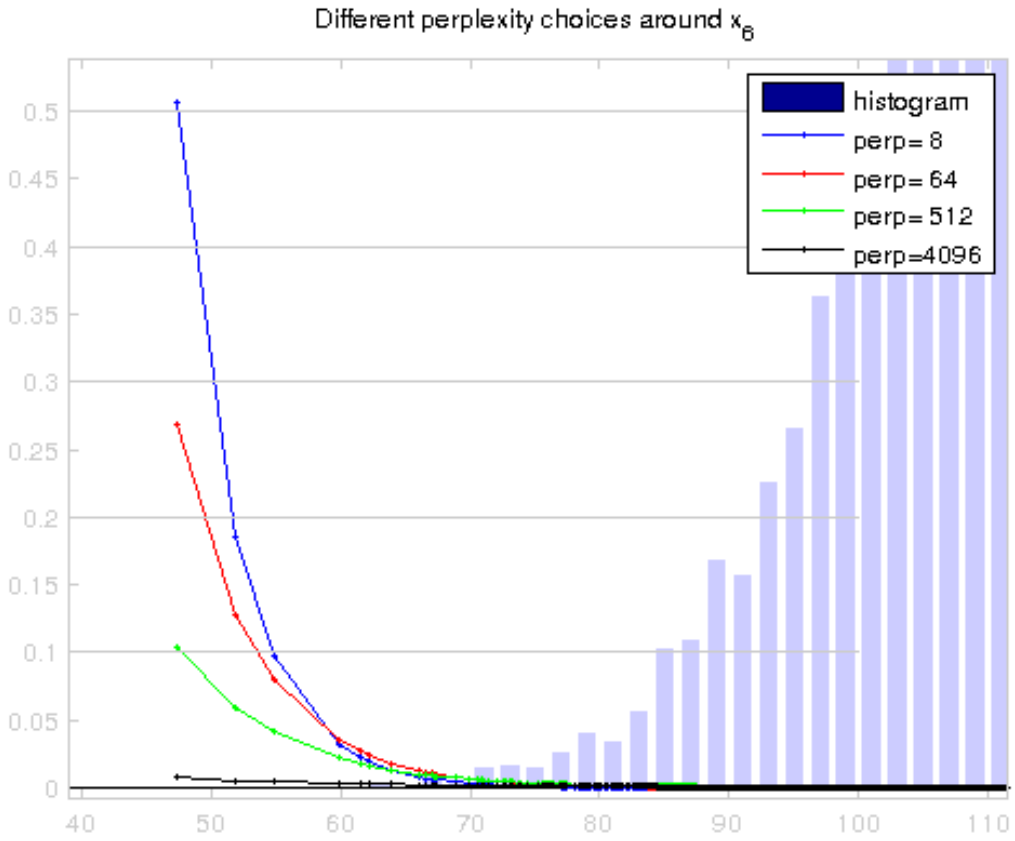

Figure 3.9: Perplexity values in relation to the histogram of the distances to point x_6 in a random selection of 7000 points from the MNIST hand-written character data set.

Chapter 4

Stochastic Discriminant Analysis

In this chapter, the new supervised dimension reduction technique for classification datasets, Stochastic Discriminant Analysis, will be introduced. Some comparisons with existing methods will be conducted.

4.1 Stochastic Discriminant Analysis

Formally, we are reducing the size of a data matrix containing n observations each with D variables (dimensions): $\mathbf{X} \in \mathbb{R}^{n \times D}$. The data vectors $\mathbf{x}_i \in \mathbb{R}^{1 \times D}$ are rows in $\mathbf{X}$. We reduce the amount of variables in $\mathbf{X}$ by finding a subspace of it by a linear projection: $\mathbf{Z} = \mathbf{XW}$, where $\mathbf{Z}$ is a $\mathbb{R}^{n \times d}$ matrix, $\mathbf{W} \in \mathbb{R}^{D \times d}$, and $d \ll D$. We are using class information from the response matrix $\mathbf{Y} = [\mathbf{y}_1; \mathbf{y}_2; ...; \mathbf{y}_n]^T \in \mathbb{I}^{n \times d_y}$ to find this projection. The response variables $\mathbf{y}_i$ are sequences of d_y binary numbers, specifying the class labels. The linear subspace is searched by matching point adjacencies (probability mass functions) in the embedding space with point adjacencies in the response space. The probabilities between points i and j in the $\mathbf{Z}$-space are:

$$q_{ij}(\mathbf{W}) = \frac{(1 + ||\mathbf{z}_i - \mathbf{z}_j||_2^2)^{-1}}{\sum_{k=1}^{n} \sum_{l=1, l \neq k}^{n} (1 + ||\mathbf{z}_k - \mathbf{z}_l||_2^2)^{-1}}, \tag{4.1}$$

where $\mathbf{z}_i = \mathbf{x}_i \mathbf{W}$ is the low-dimensional embedding coordinate. The elements q_{ij} are called t-distributed, because of the similarity with the probability density function of the t-distribution with one degree of freedom. The probabilities of the response space are $p_{ij} = \bar{p}_{ij}/\sigma$, where the normalization term $\sigma = \sum_{ij} \bar{p}_{ij}$ and

$$\bar{p}_{ij} = \begin{cases} 1, & \text{if } \mathbf{y}_i = \mathbf{y}_j \\ \epsilon, & \text{otherwise} \end{cases}, \tag{4.2}$$

where $\epsilon > 0$ is any small number. The target probabilities define *ideal* distances. By setting $\epsilon \to 0$ we essentially have a stochastic version of the LDA principle: minimize within-class ($\mathbf{y}_i = \mathbf{y}_j$) and maximize between-class ($\mathbf{y}_i \neq \mathbf{y}_j$) distances.

Figure 4.1 shows the t-distribution with one degree of freedom (before normalization). A target probability (red cross) as well as two hypothetical realized probabilities (blue dots) with their associated distances are shown. Both realized probabilities are at the same distance from the target distance, but they are invoking different offsets in the

probability (blue vertical line). As such, embedding different class coordinates too close to each other invokes a heavier penalty than embedding them too far away from each other. By encoding distances between points in the low-dimensional embedding space as t-distributed probabilities, we can discourage the embedding of class clusters too close to each other. We can write the cost function as:

$$J(\mathbf{W}) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \log \frac{p_{ij}}{q_{ij}(\mathbf{W})} + \lambda \sum_{i=1}^{D}\sum_{j=1}^{d} \mathbf{W}_{ij}^2. \tag{4.3}$$

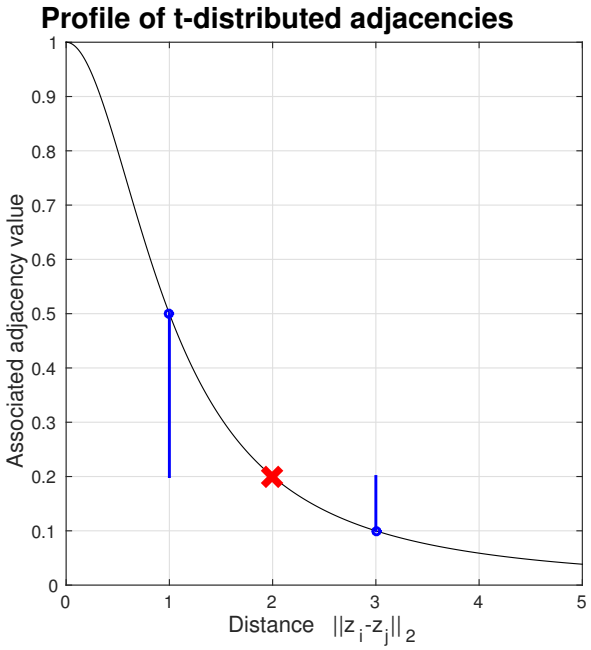

Figure 4.1: The t-distributed adjacency value is associated with a distance in the embedding coordinate $||\mathbf{z}_i - \mathbf{z}_j||_2$. The adjacency value is transformed to a probability by normalizing it. The normalization is done by dividing it with the sum of all other elements in the matrix $\mathbf{Q}$, Equation (4.1). The difference in the target probability and the realized probability is symbolized by the blue line between the realized adjacency values (blue dots) and the target (red cross).

We are searching for the thin linear projection matrix $\mathbf{W}$ that minimizes the Kullback-Leibler divergence of approximating probability distribution P with Q. The inefficiency of encoding *ideal* distances in the response space using *realized* distances in the embedding space is measured. The t-distribution causes asymmetric distance penalties: the cost function is more sensitive to deviations in within-class distances than it is to between-class distances. In visualization applications the target dimension is most often two-dimensional, in which case many methods simply extract the first 2 or 3 dimensions of the projection matrix. This is suboptimal in the presence of multiple classes. The simplex corners represent the edges of the high-dimensional shapes. Projecting the data onto a lower dimension than ν-1 means that the features are truncated to value zero, which means that these dimensions would typically be projected between the remaining simplex edges that are not truncated.

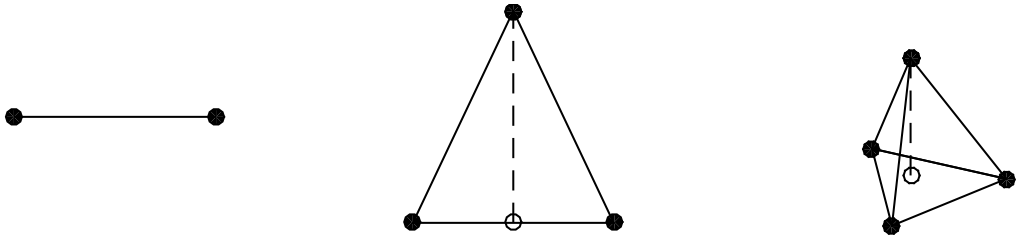

Figure 4.2: Projecting to a lower dimension directly from the optimal simplex structure is typically done by truncating the last dimensions.

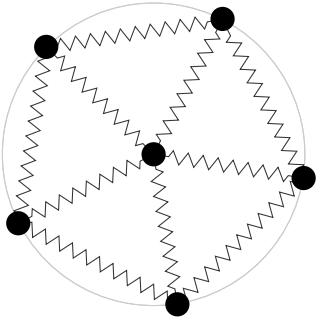

Figure 4.3: An ideal embedding of 6 classes into 2D. Same-class data points are drawn together while being separated from other class data points.

The first projections from the optimal space are illustrated in Figure 4.2. Rather than extracting the first few columns of the projection matrix optimized for a higher target dimension ν-1, the projection matrix should be optimized with respect to embedding dimensionality to maximize class discrimination. SDA optimizes the projection matrix based on the target dimensionality. Matching distances creates a regular simplex structure if the target dimension is high enough. This optimal structure is found already in (ν-1)-dimensional space, where ν is the number of classes in the dataset. If not achievable, the points are placed so as to maximize the space between classes. The KL divergence is not symmetric in its arguments and reversing the order of the arguments causes a different cost. The version of KL divergence introduced here ($\sum_{i=1}^{n}\sum_{i=1}^{n} p_{ij}\log(p_{ij}/q_{ij})$ is also called *M-projection* [45]. The M-projection is infinite if q_{ij} is zero, therefore it is called zero-avoiding. Thus, the absolute distances between cluster centers become softly bounded. This creates tension between the class clouds: classes need to be separated, but they cannot be infinitely far away because of the zero-avoiding effect. The expected effect is shown in Figure 4.3. In practice, the optimization criterion converges slowly with small values of ϵ. Therefore, we choose $\epsilon = 1/\nu$ in general. We can also use an additive Tikhonov regularization term [57]. If the value of the regularization term λ is searched by cross-validation, we refer to the method as Regularized SDA, denoted as RSDA. Normally λ is set to zero. Tikhonov regularization is often applied to *ill-posed* [1] problems. In SDA, we have local solutions where the solution depends on the initialization. The initial solution in SDA is obtained with PCA, giving orthogonal vectors with maximum variance. In high-dimensional cases, regularization can help in moving past the initialization. Additionally, the optimization process can also be made smoother by constraining the elements of $\mathbf{W}$.

4.2 Gradient

Equation (4.3) presented the cost function. The minimum of that cost function is obtained from the zeros of its gradient. In this problem, the analytic solution is intractable, so we move in the direction of the negative gradient to find the minimum iteratively (gradient descent). The iterative procedure is cut off at some tolerance, typically 10^{-5}, i.e. the decrease in the cost function value between two iterations needs to be less than 10^{-5} to stop the optimization procedure. A great deal of research has been conducted to obtain efficient methods of using the gradient descent for optimization of nonlinear problems. Some of the most important algorithms were discussed in Section 2.4.

The essential steps in obtaining the gradient are written here. We use the shorthand notation $q_{ij} = q_{ij}(\mathbf{W})$. We also write the distance in the embedding space as $D_{ij} = D_{ij}(\mathbf{W}) = ||\mathbf{z}_i(\mathbf{W}) - \mathbf{z}_j(\mathbf{W})||_2^2 = \boldsymbol{\tau}_{ij}\mathbf{W}\mathbf{W}^T\boldsymbol{\tau}_{ij}^T = (\mathbf{x}_i - \mathbf{x}_j)\mathbf{W}\mathbf{W}^T(\mathbf{x}_i - \mathbf{x}_j)^T$. The matrices $\mathbf{P}, \mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\mathbf{D}$ are $\mathbb{R}^{n\times n}$ matrices. p_{ij}, q_{ij}, $\bar{q_{ij}}$ and D_{ij} denote their elements.

$$
\begin{aligned}
\frac{dKL(\mathbf{P}||\mathbf{Q}(\mathbf{W}))}{d\mathbf{W}} &= \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\frac{1}{q_{ij}}(-1)\frac{dq_{ij}}{d\mathbf{W}} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}(-1)\Big[\sum_{k=1}^{n}\sum_{l=1}^{n} q_{kl}\bar{q}_{kl}\frac{dD_{kl}}{d\mathbf{W}} - \bar{q}_{ij}\frac{dD_{ij}}{d\mathbf{W}}\Big] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\bar{q}_{ij}\frac{dD_{ij}}{d\mathbf{W}} - \sum_{k=1}^{n}\sum_{l=1}^{n} q_{kl}\bar{q}_{kl}\frac{dD_{kl}}{d\mathbf{W}} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}(p_{ij} - q_{ij})\bar{q}_{ij}\frac{dD_{ij}}{d\mathbf{W}} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}(p_{ij} - q_{ij})\bar{q}_{ij}\boldsymbol{\tau}_{ij}^T\boldsymbol{\tau}_{ij}\mathbf{W},
\end{aligned}
\tag{4.4}
$$

since $\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}k = (\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij})k = k$, where k is an arbitrary constant. Here $(1 + D_{ij})^{-1} = \bar{q}_{ij}$ denotes the unnormalized probability. Adding the regularization term we get

$$
\frac{dJ}{d\mathbf{W}} = \sum_{i=1}^{n}\sum_{j=1}^{n}(p_{ij} - q_{ij})\bar{q}_{ij}\boldsymbol{\tau}_{ij}^T\boldsymbol{\tau}_{ij}\mathbf{W} + 2\lambda\mathbf{W}. \tag{4.5}
$$

In matrix form the expression becomes

$$
\nabla_{\mathbf{W}} J = 2\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W} + 2\lambda\mathbf{W}, \tag{4.6}
$$

where $\mathbf{L}$ is calculated as:

$$
\begin{aligned}
\mathbf{G} &= (\mathbf{P} - \mathbf{Q}) \odot \bar{\mathbf{Q}} \\
\mathbf{G}^+ &= \mathbf{G} + \mathbf{G}^T \\
\mathbf{\Lambda} &= \sum_{j=1}^{n} \mathbf{G}_{ij}^+ \\
\mathbf{L} &= \mathbf{G}^+ - \mathbf{\Lambda}
\end{aligned}
\tag{4.7}
$$

Here $\odot$ denotes the Hadamard product (element-wise multiplication) and $\mathbf{G}^+$ is a symmetrized matrix of $\mathbf{G} \in \mathbb{R}^{n \times n}$ and $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the row sum of $\mathbf{G}^+$. The matrix $\mathbf{L}$ is the difference between two Laplacian matrices $\mathbf{L} = \mathbf{L_P} - \mathbf{L_Q}$, where $\mathbf{L_P}$ is calculated from the adjacency matrices $\mathbf{G_P} = \mathbf{P} \odot \bar{\mathbf{Q}}$ and $\mathbf{G_Q} = \mathbf{Q} \odot \bar{\mathbf{Q}}$. A Laplacian matrix is a symmetric diagonally dominant matrix and therefore positive definite, however $\mathbf{L}$ need not be positive semi-definite.

Algorithm 7 shows pseudo-code for obtaining a projection matrix $\mathbf{W}$ with SDA. The projection matrix $\mathbf{W}$ and its gradient need to be vectorized before passing them to the optimization algorithm. The vectorized projection matrix $\mathbf{w}$ and its vectorized gradient $\mathbf{g}$ can be plugged to any gradient-based optimization method discussed in Section 2.4. The optimization methods determine the descent direction and a line search method determines the step length. The optimization and line search methods might require additional function evaluations. At the end, the search directions are orthogonalized with SVD. The usage of thin SVD saves computational time.

Note that the target probabilities p_{ij} are determined based on the labeling of the elements in the beginnning of the algorithm, but the model probabilities q_{ij} depend on the low-dimensional coordinates and need to be recalculated at each iteration.

Algorithm 7: Gradient-based minimization for SDA

Input: Input matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$ and response matrix $\mathbf{Y} \in \mathbb{I}^{n \times d_y}$, initial projection matrix $\mathbf{W}_0 \in \mathbb{R}^{D \times d}$, regularization term λ and optimality tolerance δ
Output: Projection matrix $\mathbf{W}$
1. Calculate target probabilities $\mathbf{P} \in \mathbb{R}^{n \times n}$ with Equation (4.2)
2. Assign $\mathbf{W} = \mathbf{W}_0$
3. Calculate model probabilities $\mathbf{Q}$ with Equation (4.1)
4. Calculate cost $C_0 = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{P}_{ij} \log(\mathbf{P}_{ij}/\mathbf{Q}_{ij}) + \lambda \sum_{i=1}^{D} \sum_{i=1}^{d} \mathbf{W}_{ij}$
5. Assign $t = 0$, $\delta_C = \infty$
while $\delta_C > \delta$ **do**
6. Compute gradient $\nabla_{\mathbf{W}} J_0 = \mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W} + \lambda\mathbf{W}$
7. Vectorize projection matrix, $\mathbf{w}_t = \text{vec}(\mathbf{W})$
8. Vectorize gradient, $\mathbf{g}_t = \text{vec}(\nabla_{\mathbf{W}} J)$
9. *Determine descent direction* $\boldsymbol{d}_t$
10. *Determine step length* η_t
11. Update solution vector $\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t \boldsymbol{d}_t$
12. Reshape the vector $\mathbf{w}_t$ into matrix $\mathbf{W}_t$
13. $t = t + 1$
14. Update model probabilities $\mathbf{Q}$ with Equation (4.1)
15. Calculate new cost C_t with Equation (4.3)
16. Update change in cost $\delta_C = C_t - C_{t-1}$
end while
17. Orthogonalize $\mathbf{W}_t$ with thin SVD: $\hat{\mathbf{U}}\hat{\mathbf{S}}\mathbf{V} = \mathbf{W}_t$
18. Return $\mathbf{W} = \hat{\mathbf{U}}\hat{\mathbf{S}}$

The evaluation of the gradient is the most time-consuming part of the optimization. The gradient in matrix form is $2\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W}$, but it can be calculated in two ways: multiplying the matrices from the left side or from the right side. The computation environment

Matlab will automatically calculate the matrices from the left side. The computational cost of multiplying a $n \times p$ matrix with a $p \times m$ matrix is $O(npm)$. We can compare the computational complexity of multiplying from either side. The size of each matrix is shown below in Table 4.1.

Table 4.1: The size of each matrix in the gradient expression.

Matrix	Size
$\mathbf{X}^T$	$D \times n$
$\mathbf{L}$	$n \times n$
$\mathbf{X}$	$n \times D$
$\mathbf{W}$	$D \times d$

Table 4.2: Computational costs associated with evaluating the gradient when doing the matrix multiplication from the left side.

Left side multiplication	Size	Operations
$\mathbf{X}^T\mathbf{L}$	$D \times n \times n \times n$	$O(Dn^2)$
$(\mathbf{X}^T\mathbf{L})\mathbf{X}$	$D \times n \times n \times D$	$O(D^2n)$
$(\mathbf{X}^T\mathbf{L}\mathbf{X})\mathbf{W}$	$D \times D \times D \times d$	$O(D^2d)$

Table 4.3: Computational costs associated with evaluating the gradient when doing the matrix multiplication from the right side.

Right side multiplication	Size	Operations
$\mathbf{X}\mathbf{W}$	$n \times D \times D \times d$	$O(Dnd)$
$\mathbf{L}(\mathbf{X}\mathbf{W})$	$n \times n \times n \times d$	$O(n^2d)$
$\mathbf{X}^T(\mathbf{L}\mathbf{X}\mathbf{W})$	$D \times n \times n \times d$	$O(Dnd)$

The costs for the individual operations in multiplying from left is summarized in Table 4.2 and from the right in Table 4.3. In total, the cost for evaluating the matrix multiplication multiplying from the left is $O(Dn^2 + D^2n + D^2d)$. Multiplying from the right side we get $O(nDd + n^2d)$, since the constant can be ignored. In our case $d \ll D$, since our application is specifically dimension reduction. Multiplying from the right side would be faster in most cases.

Both versions were run on the learning sets of Olivetti ($n = 266, D = 4096$), Phoneme ($n = 3005, D = 256$), USPS ($n = 6198, D = 256$) and COIL-20 ($n = 1224, D = 16384$) data sets, and the dimensionality was reduced to 2D. The LBFGS algorithm was used in all cases. The required convergence times are summarized in Table 4.4. The speed-up in the very high-dimensional Olivetti and COIL-20 data set was considerable, approximately 82% and 96%. The computation time reduction in the two large data sets were approximately 10%.

Table 4.4: Total computation time of computing the transformation matrix in SDA using the LFBGS algorithm on different data sets. Only the direction of the matrix multiplication was changed between the optimization runs.

Matrix	Left-multiplication	Right-multiplication
Olivetti	47.17s	8.35s
Phoneme	48.10s	44.30s
USPS	1116s	1006s
COIL-20	2608.88s	106.98s

4.3 Hessian

The Hessian can be directly obtained from the gradient as $\mathbf{H} = 2\mathbf{X}^T\mathbf{L}\mathbf{X}$, however a closer inspection of the Hessian is in order. This matrix $\mathbf{L}$ was calculated in Equation (4.7). The matrix is the difference between two Laplacian matrices $\mathbf{L} = \mathbf{L_P} - \mathbf{L_Q}$, where $\mathbf{L_P}$ is calculated from the adjacency matrix $\mathbf{G_P} = \mathbf{P} \odot \bar{\mathbf{Q}}$ and $\mathbf{L_Q}$ from $\mathbf{G_Q} = \mathbf{Q} \odot \bar{\mathbf{Q}}$. A Laplacian matrix is a diagonally dominant matrix and therefore always positive definite. $\mathbf{L}$ is not guaranteed to be positive semi-definite because it is the difference between two psd matrices. The use of matrix $\mathbf{H}^+ = 2\mathbf{X}^T\mathbf{L_P}\mathbf{X}$ in place of the real Hessian is called Spectral Direction optimization. Using the partial Hessian $\mathbf{H}^+$ instead of the true Hessian $\mathbf{H}$, we can guarantee that we always proceed in a descent direction. Algorithm 7 can be extended to use Hessian information with a small modification. The modified algorithm is shown as Algorithm 8.

If the Hessian information is used, the computational load per iteration is larger. The evaluation of the Hessian has the computational complexity $O(D^2n + Dn^2)$. The inverse Hessian can be calculated by Cholesky factorizing the Hessian $O(D^3)$ and then doing two backsolves $O(D^2)$ [2]. Thus, the computational complexity for evaluating a direction using the Hessian is $O(Dn^2 + n^3)$. The use of the partial Hessian might reduce the total amount of function evaluations required because of the more informed calculation of the direction. The individual contributions to the time complexity are shown in Table 4.5. The total time complexity for one iteration, including the right-side calculation of the gradient is $O(nDd + n^2d + Dn^2 + D^2n + D^3)$. The time complexity scales with the third power of the input dimension, making it unattractive in very high-dimensional problems.

Algorithm 8: Spectral gradient minimization for SDA

Input: Input matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$ and response matrix $\mathbf{Y} \in \mathbb{I}^{n \times d_y}$, initial projection matrix $\mathbf{W}_0 \in \mathbb{R}^{D \times d}$, regularization term λ and optimality tolerance δ
Output: Projection matrix $\mathbf{W}$
1. Calculate target probabilities $\mathbf{P} \in \mathbb{R}^{n \times n}$ with Equation (4.2)
2. Assign $\mathbf{W} = \mathbf{W}_0$
3. Calculate model probabilities $\mathbf{Q}$ with Equation (4.1)
4. Calculate cost $C_0 = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{P}_{ij} \log(\mathbf{P}_{ij}/\mathbf{Q}_{ij}) + \lambda \sum_{i=1}^{D} \sum_{i=1}^{d} \mathbf{W}_{ij}$
5. Assign $t = 0$, $\delta_C = \infty$
while $\delta_C > \delta$ **do**
6. Compute gradient $\mathbf{\Gamma}_{\mathbf{W}_t} = \mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W} + \lambda\mathbf{W}$
7. Compute partial Hessian $\mathbf{H}_+ = 2\mathbf{X}^T\mathbf{L}_+\mathbf{X} + 2\lambda\mathbf{I}$
8. Cholesky-decompose $\mathbf{R}^T\mathbf{R} = \mathbf{H}^+$
9. Do two backsolves to get $\boldsymbol{\nu}_t$: $\mathbf{R}^T(\mathbf{R}\boldsymbol{\nu}_t) = \mathbf{\Gamma}_{\mathbf{W}_t}$
10. Vectorize projection matrix, $\mathbf{w}_t = \text{vec}(\mathbf{W})$
11. Vectorize descent direction, $\mathbf{d}_t = \text{vec}(\boldsymbol{\nu})$
12. Find step length η_t that satisfies the Wolfe conditions
13. Update solution vector $\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t \boldsymbol{d}_t$
14. Reshape the vector $\mathbf{w}_t$ into matrix $\mathbf{W}_t$
15. $t = t + 1$
16. Update model probabilities $\mathbf{Q}$ with Equation (4.1)
17. Calculate new cost C_t with Equation (4.3)
18. Update change in cost $\delta_C = C_t - C_{t-1}$
end while
19. Orthogonalize $\mathbf{W}_t$ with thin SVD: $\hat{\mathbf{U}}\hat{\mathbf{S}}\mathbf{V} = \mathbf{W}_t$
20. Return $\mathbf{W} = \hat{\mathbf{U}}\hat{\mathbf{S}}$

4.4 A theoretical connection between the cost function in SDPP and SDA

We can think of the cost functions in SDPP and SDA in a slightly different way. In SDPP we were considering squared Euclidean distances (SED) as the random variables that we were trying to predict. The loss function was also the squared Euclidean distance. Here, we will instead consider probabilities calculated from squared distances as the random variables and use the Kullback-Leibler divergence (KLD) as the cost function. In essence, we are changing from a scenario of matching squared distances to a scenario of matching probability distributions.

Both SED and KLD are part of a broader concept, the Bregman divergences [7, 54]. These divergences are defined by having a strictly convex function $\phi(x_i)$ that defines the associated loss function as:

$$D_\phi(x_i, y_i) = \phi(x_i) - \phi(y_i) - \langle x_i - y_i, \nabla\phi(y_i)\rangle,$$

where $\langle \cdot, \cdot \rangle$ refers to the dot product. By setting $d_i \to x_i$, where d_i is the vectorized squared distance in the low-dimensional space, $N = n^2$, setting y_i to be the vectorized

Table 4.5: Computational costs associated with evaluating the partial Hessian direction. **a** is a temporary vector.

Left side multiplication	Operation	Computational complexity
$\mathbf{X}^T\mathbf{L}$	Matrix multiplication	$O(Dn^2)$
$(\mathbf{X}^T\mathbf{L})\mathbf{X}$	Matrix multiplication	$O(D^2n)$
$\mathbf{R}\mathbf{R}^T = (\mathbf{X}^T\mathbf{L}\mathbf{X})$	Cholesky factorization	$O(D^3)$
$\mathbf{a} = -\mathbf{R}^T/\mathbf{g}$	Backsolve	$O(D^2)$
$\mathbf{d} = -\mathbf{R}/\mathbf{a}$	Backsolve	$O(D^2)$

squared distances in the response space and $\phi(x_i) = ||x_i||^2$ (L^2-norm of the point-to-point distances in low-dimensional space), we get the SDPP cost function. By setting the data points x_i to be the vectorized probability values $q(d_i)$, y_i to be the vectorized target probabilities p_i and by setting $\phi(x_i) = x_i \log x_i$ (Shannon entropy of the distribution of the distances in the low-dimensional space), the loss function is the Kullback-Leibler divergence, as used in Stochastic Discriminant Analysis.

Table 4.6: Comparison of squared Euclidean and Kullback-Leibler divergences. x_i is the random variable that we are in general trying to predict and y_i is its estimate.

Divergence	Squared Euclidean	Kullback-Leibler
Domain of the random variable	$x_i \in \mathbb{R}^N$	$x_i \in$ N-simplex
Related strictly convex function $\phi(x)$	$\sum_{i=1}^N ||x_i||^2$	$\sum_{i=1}^N x_i \log x_i$
Divergence $D_\phi(x, y)$	$\sum_{i=1}^N ||x_i - y_i||^2$	$\sum_{i=1}^N x_i \log(x_i/y_i)$

Differences between SED and KLD are summarized in Table 4.6. The N-simplex refers to the geometrical concept of the non-negative probability simplex spanned by $\sum_{i=1}^N x_i = 1$.

4.5 Embedding with the Iris dataset

The SDA embedding of the Iris dataset can be seen in Figure 4.4. The target probabilities **P** and the realized model probabilities **Q** are also shown. The target and model probabilities are in the same range. The optimal embedding in Figure 4.5 is not achieved. The Iris dataset contains too little variability in the input variables to achieve the simplex structure. We can see that the third and second class are encoded too close to each other by comparing the target probabilities and the model probabilities. The second class is a bit too close to the first class (shade too dark) and the third class is a bit too far away from the first class (shade too light). The classes are separated quite well.

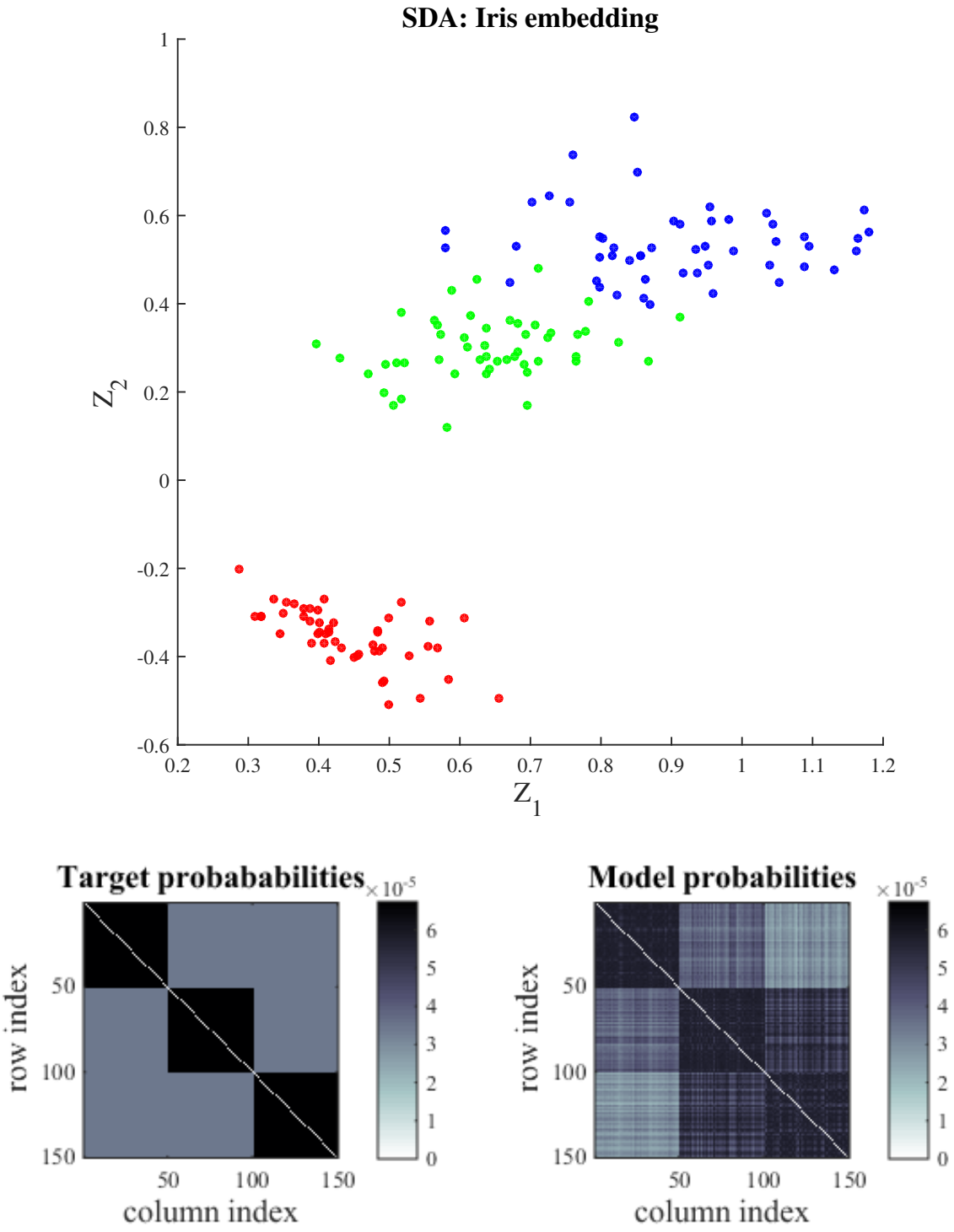

Figure 4.4: The SDA projection of the Iris data set onto 2D.

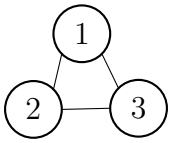

Figure 4.5: The optimal structure for a 2D embedding of the Iris dataset.

Chapter 5

Datasets

In this chapter all the datasets used in the experiments in this thesis are introduced. They are shown in Table 5.1. These datasets are widely used to test dimensionality reduction methods. The datasets are mostly multiclass datasets, and half of the datasets contain at least ten classes.

Table 5.1: Data sets used in this paper.

Data set	Samples	Variables	Classes
Iris	150	4	3
Wisconsin Breast Cancer	683	9	2
Wine	178	13	3
USPS	9298	256	10
MNIST5k	5000	784	10
Phoneme	4509	256	5
Olivetti faces	400	4096	40
COIL-20	1440	16384	20
COIL-100	7200	16384	100

First in the list are the oldest datasets. The Iris dataset, the Wisconsin Breast Cancer dataset and the Wine dataset were gathered during the last century and they contain few classes, few variables and few data points. In the Iris dataset, morphological varieties of the Iris flower are identified by quantitative measurements of flowers. In the Wine dataset, wine species are identified based on chemical test results. In the Wisconsin Breast Cancer dataset, tumors are classified as benign or malignant based on physical measurements. The datasets contain too many dimensions for visual inspection (only two or three variables can be viewed simultanously), but the dynamics become easy to understand after dimension reduction. As such, these datasets are often used to demonstrate that a particular dimension reduction technique works. The datasets are retrieved from the UCI machine learning database [5].

The three following datasets contain more elements. The USPS and MNIST datasets contain phographed or scanned hand-written grey-scale digits. The analysis of these datasets lead to the automatic number recognition at post offices. The k-nearest neighbors classification of digits is despite its simplicity still among the favored algorithms for digits classification. The 1-$\mathcal{NN}$ error rate is about 3.8% on the USPS dataset, compared to the human error rate of about 2.5% in the USPS dataset [42]. Deep learning algorithms

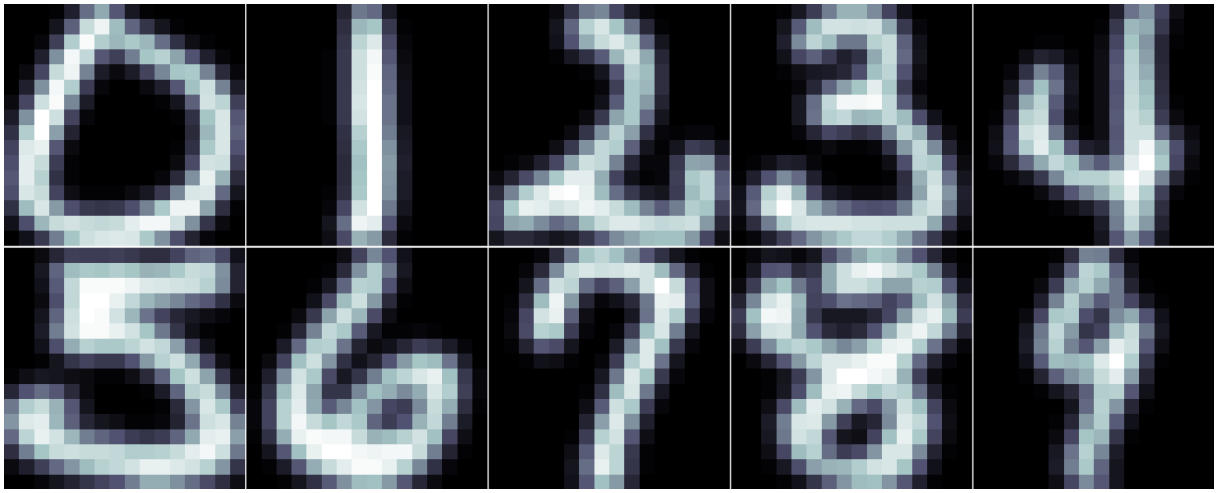

Figure 5.1: Examples of digits from the USPS dataset. The images are the first digits for each class.

have achieved error rates of about 1% in the MNIST dataset [39]. Figure 5.1 shows some example digits from the USPS dataset. The digits often occupy the same area of the pixel space. The phoneme dataset contains five different sounds [28]: three vowel pronunciations 'aa' (the *a* in dark, IPA: ɑ), 'ao' (the *a* in water, IPA: ɐ) and 'iy' (the 'e' in she: IPA: i) as well as two consonants 'dcl' (*d* in dark) and 'sh' (*sh* in she). The 'aa' and 'ao' sounds are very similar and difficult to separate linearly.

The last three datasets are high-dimensional datasets. The Olivetti faces dataset (sometimes called ORL / AT&T) [55] contains 40 persons photographed in 10 pictures each. Each sample is a 64-by-64 pixel image, giving 4096 variables. Faces in general contain similar objects (faces). Subjects in different classes look similar: their eyes, noses etc. are typically centered to the same position. The dataset is challenging for pixel-intensity based recognition because the faces are irregularly varying: the camera angle might be different or the subject might have different facial expressions in the images. Figure 5.2 shows some example pictures from the data set.

The Columbia Object Image Library (COIL-20) contains rotating images of 20 objects, photographed at 5 degree intervals [48]. The images are 128-by-128 pixel grey-scale images. Images include objects such as rubber ducks, toy cars and jars. In total, there are 1440 samples in 16384 dimensions. The objects are mostly dissimilar and occupy different positions in the pixel space, which makes it easy to recognize objects based on pixel-intensity. Figure 5.3 shows a representative image for each class from the COIL-20 dataset. To give an example, there are three classes of different cars. COIL-100 [46] is a larger dataset of COIL-20, containing 100 objects in RGB colors. COIL-100 is more difficult because it contains several images of the same type, for example nine classes of different cars. In this thesis the COIL-100 dataset is transformed to grayscale, giving 16384 dimensions. The variables in all datasets are normalized to have zero mean and unit variance.

Figure 5.2: Examples of faces from the Olivetti data set. The rows shows four randomly picked standardized images of the 5 different persons.

Figure 5.3: Examples objects from the COIL-20 dataset. The images are the first pictures of each class.

Chapter 6

Evaluation

In this chapter the proposed method Stochastic Discriminant Analysis is evaluated experimentally. The computational environment is described in Section 6.1. The experimental evaluation is divided into two parts. First, three case studies on different datasets are conducted in Sections 6.2, 6.3 and 6.4. The three datasets are multiclass high-dimensional image datasets: Olivetti faces, USPS and COIL-20, introduced in Chapter 5. The Olivetti faces dataset (6.2) contains photographies of people. The input dimensionality is very high. The USPS dataset (6.3) contains a large number of hand-written digits in ten classes with a smaller input dimension. COIL-20 (6.4) features 20 very high-dimensional images of rotating objects photographed at fixed angle intervals. In these case studies, the classification accuracies for a range of target dimensionalities are calculated and compared with other methods. The two-dimensional projections with learning points and out-of-sample points are shown. A regularization parameter search scheme for SDA is described in Section 6.2 and the runtime with different optimization algorithms on these datasets is determined in Section 6.5. In Section 6.6, a comparison of the out-of-sample classification accuracies on two-dimensional projections with state-of-the-art methods is conducted over a range of datasets. The utilized datasets have been summarized in Table 5.1 in Chapter 5.

6.1 Computational environment

In this section I will discuss the environment used in the calculations. The MATLAB computer software is used for scientific computation in this thesis. The computations are mainly performed on Ubuntu Linux 14.04 with a 4-core Intel i5-4570 @ 3.2 GHz processor and 16GB RAM. Some experiments were calculated in the Triton environment, running Scientific Linux with either Intel Xeon X5650 @ 2.67GHz processors or AMD Opteron 2435 @ 2.6GHz processors, with a variable amount of memory. The memory requirements for running these experiments were roughly in the range 2GB to 16GB. The time required to find an embedding ranged from seconds (Olivetti dataset) to hours (COIL-100 dataset).

This thesis involves nonlinear unconstrained optimization problems. The default solvers in MATLAB are outdated, but there are some high-performance third-party optimization packages available. One of them is the nonlinear conjugate gradient function *minimize.m* by C.E. Rasmussen [52, 53]. The functions in the *minfunc*-package by M. Schmidt are equally good or faster [56]. The conjugate gradient algorithm in minimize.m uses Polak-Ribière's updating rule and a line search using quadratic and cubic polynomial

approximations. It uses the Wolfe-Powell stopping criteria and other checks to ensure that exploration is being done and that the exploration is not unboundedly large. The minfunc routines utilize functions written in C through the use of mex-files. The default algorithm in the minfunc conjugate gradient algorithm uses the Hestenes-Stiefel update formula, but there are many other options too.

We will define the hyperparameters used in various methods here. Our proposed method SDA is initialized with the PCA projection as its initial solution is all tests. In SPCA, we chose the delta kernel [9] for the response space. In the kernel version of SPCA, we selected the delta kernel for response space and a Gaussian kernel for the input space, setting the width of the Gaussian to the median value of the squared interpoint distances. gKDR was run in the partitioning mode (v) to reduce its memory requirements.

6.2 The Olivetti faces dataset

In our tests, two thirds of the data vectors were randomly selected as training vectors and one third as test vectors. The random selection was repeated ten times to average the performance and acquire error bars.

6.2.1 Two-dimensional embeddings

The Olivetti dataset was projected to two dimensions in Figure 6.1. Each data point is represented by the corresponding image of the person. Learning points do not have boxes around the images, while test points have blue or red boxes. The color depends of whether the classification based on the nearest neighbor from the learning set was correct or not. We can see that there appears to be comparatively few learning points compared to the number of test points. This is because the learning points have actually collapsed on top of each other, at zero distance. Out-of-sample data points are embedded only roughly in the right area. The data is sparse, which typically can cause overutilization of certain pixels. We can restrict the utilization of any particular pixels, putting more focus on groups of pixels. This can be achieved by regularizing SDA (RSDA). The regularization scheme is described in subsection 6.2.2.

A two-dimensional RSDA embedding of the Olivetti faces is shown Figure 6.2. The correct classifications and misclassifications have been highlighted in the figure. The learning and testing subsets are the same as in the unregularized version. The learning points are no longer projected right on top of each other: the Tikhonov regularization makes such an embedding costly. The test data points are spread less around the learning points, increasing the classification accuracy. Some of the test data points were projected too close to another class because of new facial expressions. We can see for example that the face projected at coordinates $(3, 6)$ should in fact have been projected at coordinates $(0.5, 4.5)$. The person did not have a similar facial expression as in the learning data set: we can see that the eyes were facing upwards.

The same SDA and RSDA embeddings are compared against embeddings obtained with other methods in Figure 6.3. All the figures show the 267 learning points and 133 test points for the same permutation. The embeddings are very different. In SDA, all data points belonging to the same class have collapsed on top of each other. The data clusters are roughly evenly spread over a circular area. The test points are often, but not

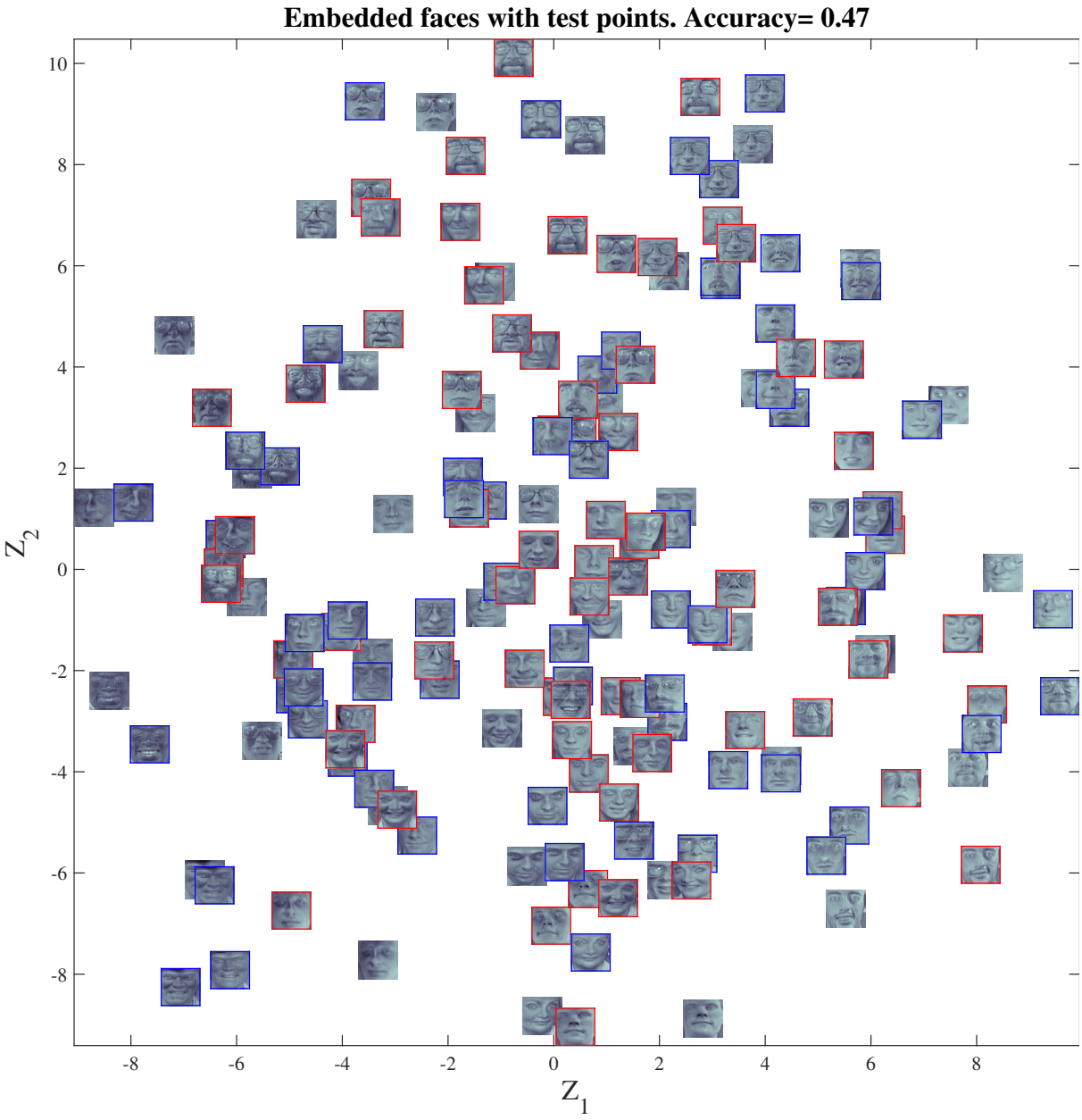

Figure 6.1: **An SDA linear embedding of the Olivetti faces dataset.** Colored borders denote projected test points. Red borders denote a misclassification, while blue borders denote a correct classification.

always, close to their own cluster centers. RSDA constrains the hypothesis made by SDA with Tikhonov regularization, making it more difficult to collapse the same class elements on each other. The test points are clearly projected closer to their own groups. LDA and SDPP were applied on a 100-dimensional PCA projection. LDA separates a few groups nicely, but many classes are inseparably mixed near origin. For the rest of the methods, the task is too demanding. The assumptions that the data is sampled frequently enough are violated in SDPP. The data dimensionality is large and LPP performs poorly. Many of the methods work extremely well on higher projection dimensions, but the methods often have problems in projecting out-of-sample test points close to their own classes in

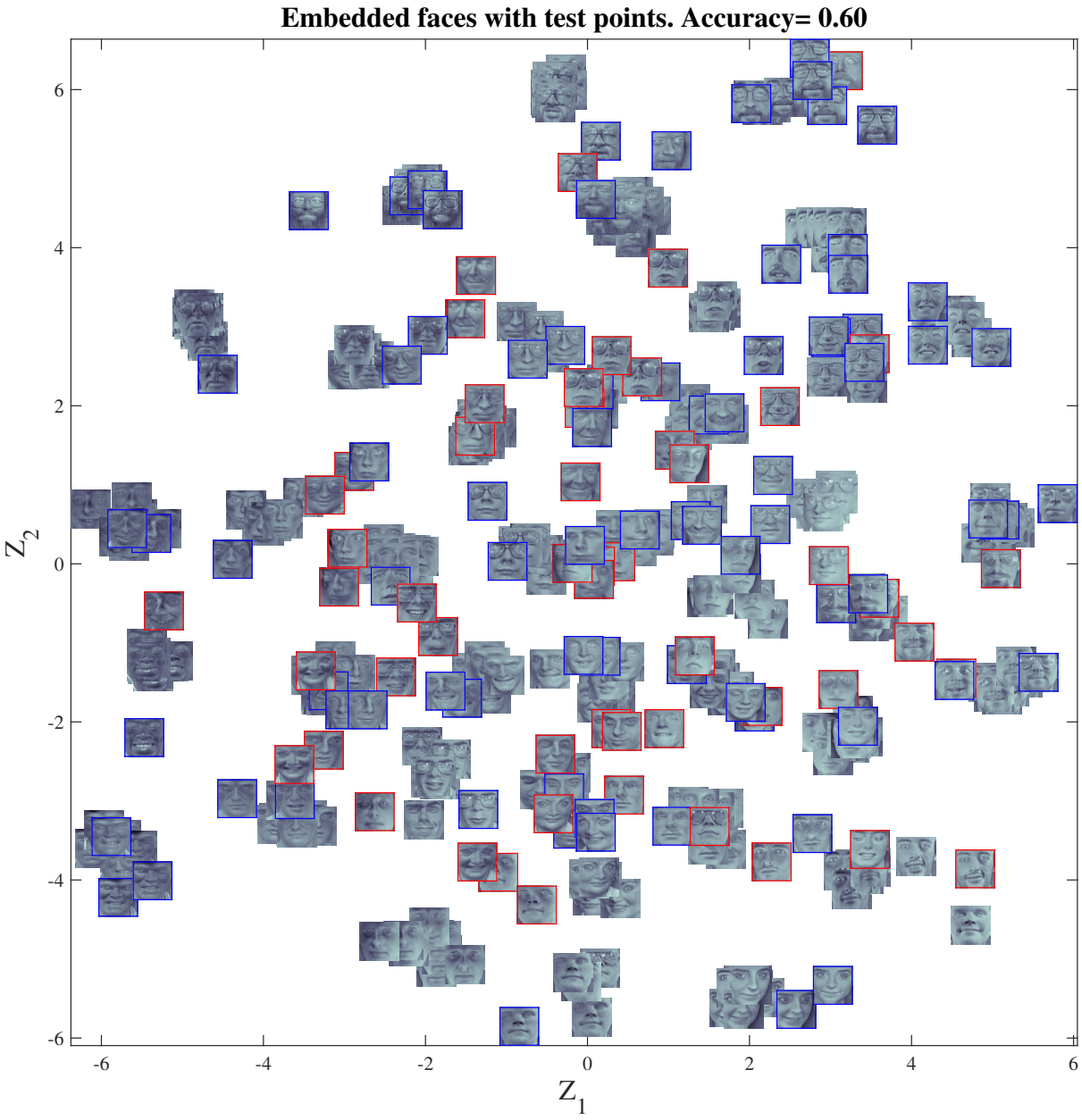

Figure 6.2: **A representative RSDA linear embedding of the Olivetti faces dataset.** Colored borders denote projected test points. Red borders denote a misclassification, while blue borders denote a correct classification.

extremely low dimensional projections.

6.2.2 Regularization parameter search

In the Olivetti dataset, Tikhonov regularization was used to guide the optimization process. The appropriate amount of regularization was searched by cross-validation. A random selection of 80% of the learning subset (213 points) was used for training and 20% were used for cross-validation (54 points). The best value for the regularization term is searched by trying six logarithmically intervalled values of λ from 10^2 to 10^{-8}. Then

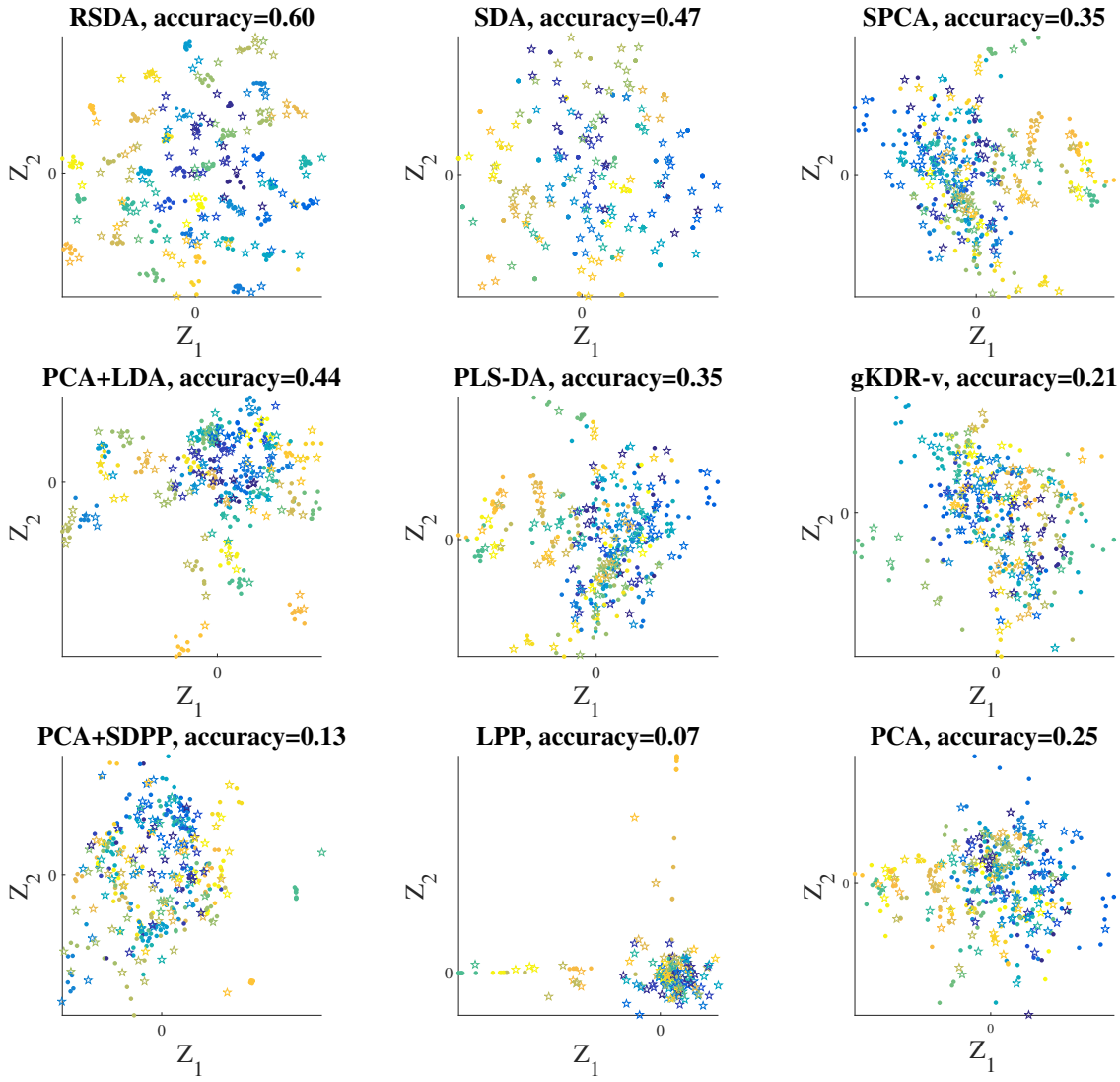

Figure 6.3: **Linear embeddings of the Olivetti faces dataset.** Dots denote projected learning points. Stars denote projected test points. The 1-$\mathcal{NN}$ classification accuracy resulting from this embedding is added to the title.

two magnification rounds were performed. To give an example, let us say that λ_1 was the regularization value that gave the smallest 1-$\mathcal{NN}$ classification error among the six tested values of regularization. The search is then magnified in the area around λ_1 by including values $10\lambda_1$ and $0.1\lambda_1$. The same criteria is used to evaluate these two new values. The smallest λ_2 giving the smallest classification error is then the subject of a magnified search with values $10^{0.5}\lambda_2$ and $10^{-0.5}\lambda_2$. In total, ten regularization values are explored in the cross-validation search. Among these values, the one that gives the smallest 1-$\mathcal{NN}$ classification error is called λ^*, which is the regularization value used for training the data.

Figure 6.4 shows one regularization search procedure. The classification error is plotted against the logarithm of the regularization term. The dimension reduction in the figure is to two dimensions. We can observe that the search is magnified twice in the region $\lambda = 10^0$. Finally, the 1-$\mathcal{NN}$ classification error on the cross-validation dataset was found to be the smallest when $\lambda = 10^{-0.5} \approx 0.32$. This search was conducted until no progress

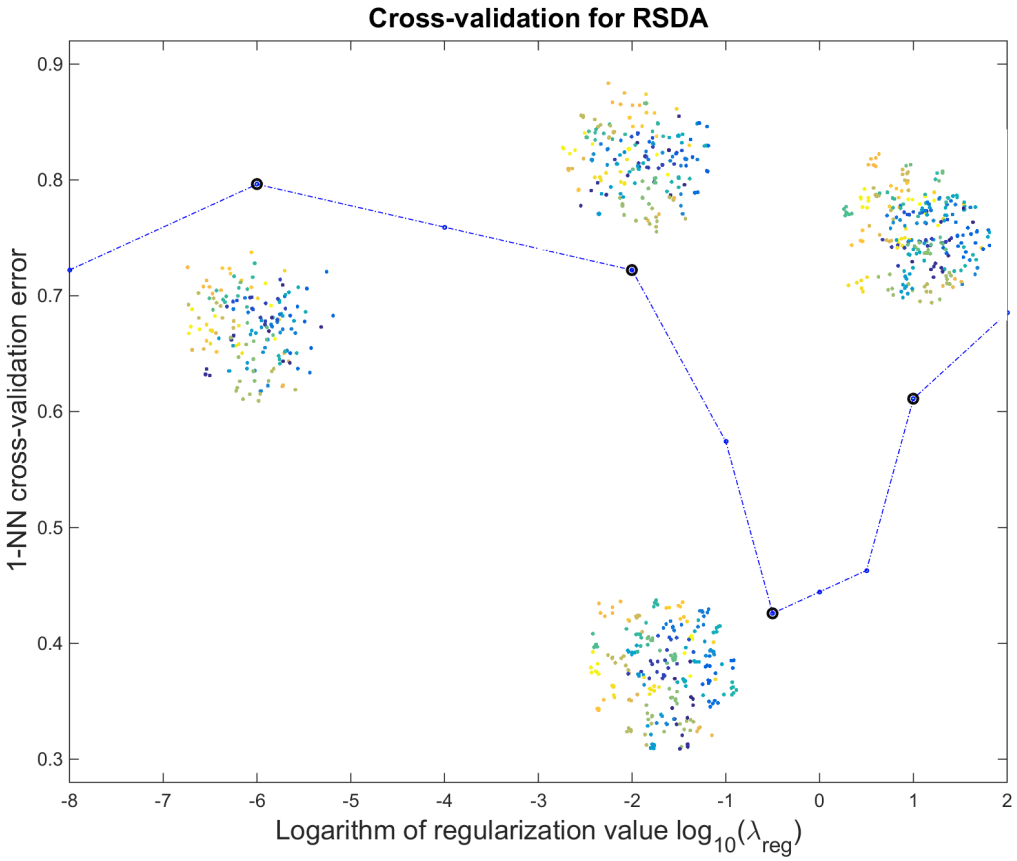

Figure 6.4: **Tikhonov regularization parameter search.** The two-dimensional embedding of the learning points are displayed for selected values of λ. The transitions between test points are quantized because of the low number of test points in the cross-validation set (54).

could be made, evaluated at a tolerance 10^{-4}. The image shows typically a U-shape that is typical in bias-variance trade-offs: too little bias (too small values of regularization) increase the variance, while too much bias (too much regularization) is not good either. The search procedure was fast, requiring less than ten seconds per value explored (typically about 3-4 seconds)[1]. The tolerance for optimality in the main algorithm was set at 10^{-5}. The SDA projection found this way is called RSDA.

6.2.3 Comparative performance over a range of dimensions

Figure 6.5 shows the classification accuracies for many dimension reduction methods for a range of target dimensions. The error bars report the mean and standard errors. The regularized algorithm shows the best performance here. The mean accuracy is highest among the methods and the error bars are among the narrowest. The method stabilizes at 98.0% 1-$\mathcal{NN}$ classification accuracy at 10D, above the 90.1% accuracy for using the whole input space. The mean classification accuracies are shown in Table 6.1 too. RSDA

[1]Run on 4-core Intel i5-4570 CPU @ 3.20GHz

Table 6.1: Mean classification accuracies with an 1-$\mathcal{NN}$ classifier a range of reduced spaces acquired with different DR methods. The tests were repeated ten times. The best results are bold-faced.

Dim.	SDA	RSDA	PCA	PLS-DA	SPCA	gKDR-v	LDA
$2D$	39	**56**	29	33	32	22	45
$3D$	59	**80**	46	51	54	38	64
$4D$	70	**89**	62	65	66	49	76
$5D$	78	**94**	70	72	73	61	83
$6D$	85	**97**	76	77	78	70	86
$8D$	91	**97**	81	85	84	83	91
$10D$	94	**98**	84	89	86	90	94
$14D$	95	**98**	86	93	89	94	95
$18D$	96	**98**	88	94	90	96	96
$22D$	96	**98**	89	94	91	96	96
$26D$	96	**98**	89	95	91	96	96
$30D$	96	**98**	89	94	91	95	97

and SDA find some of the most meaningful linear subspaces in the data.

6.2.4 Prototype Faces

In the Background Section 2.2, fisherfaces and eigenfaces were discussed. Naturally, these kind of faces can be processed using RSDA projections too. These faces will be referred to as prototype faces in this thesis. The five first orthogonal prototype faces are shown in Figure 6.6. Using the five faces alone to classify out-of-sample elements resulted in a 94% classification accuracy on its test test. In the prototype faces, black pixels indicate negative values and white pixels indicate positive values. Red pixels are almost zeros. Both negative and positive values of the prototype faces are used in obtaining low-dimensional embeddings of samples. The fisher- and eigenfaces (LDA and PCA) are shown in Figures 6.7 and (6.8). All faces are obtained by reprojecting the projection matrix into the original space, using the same seletion of learning samples.

The eigenfaces are undoubtedly the easiest to interpret. We can interpret a linear scale in the darkness of the face in eigenface 1, a left-right orientation scale on eigenface 2 (telling if the head turned in either direction) and eye shades in eigenface 5. Eigenface 1 has also caught the background lighting of the image. The first fisherface has dark eyes and a white chin, apparently a combination important for discriminating people. Negative values of the first fisherface would create different shades of dark chins, typical for bearded persons. Many of the RSDA faces feature eyes. The eye shapes appear important in discerning people. The face area is in general more red than in fisherfaces, containing more almost-zero values. We can for example see that the fifth prototype face has a "white beard" and the second face contains dark eyebrows.

We can take a look at what the most important prototype faces describe for a chosen subject. Figure 6.9 shows five pictures of subject 10 from the Olivetti dataset. Table 6.2 shows the contributions of each prototype face to the embedding location of the subject. *Projection vector* refers to the prototype faces. The subject has dark eye-circles and

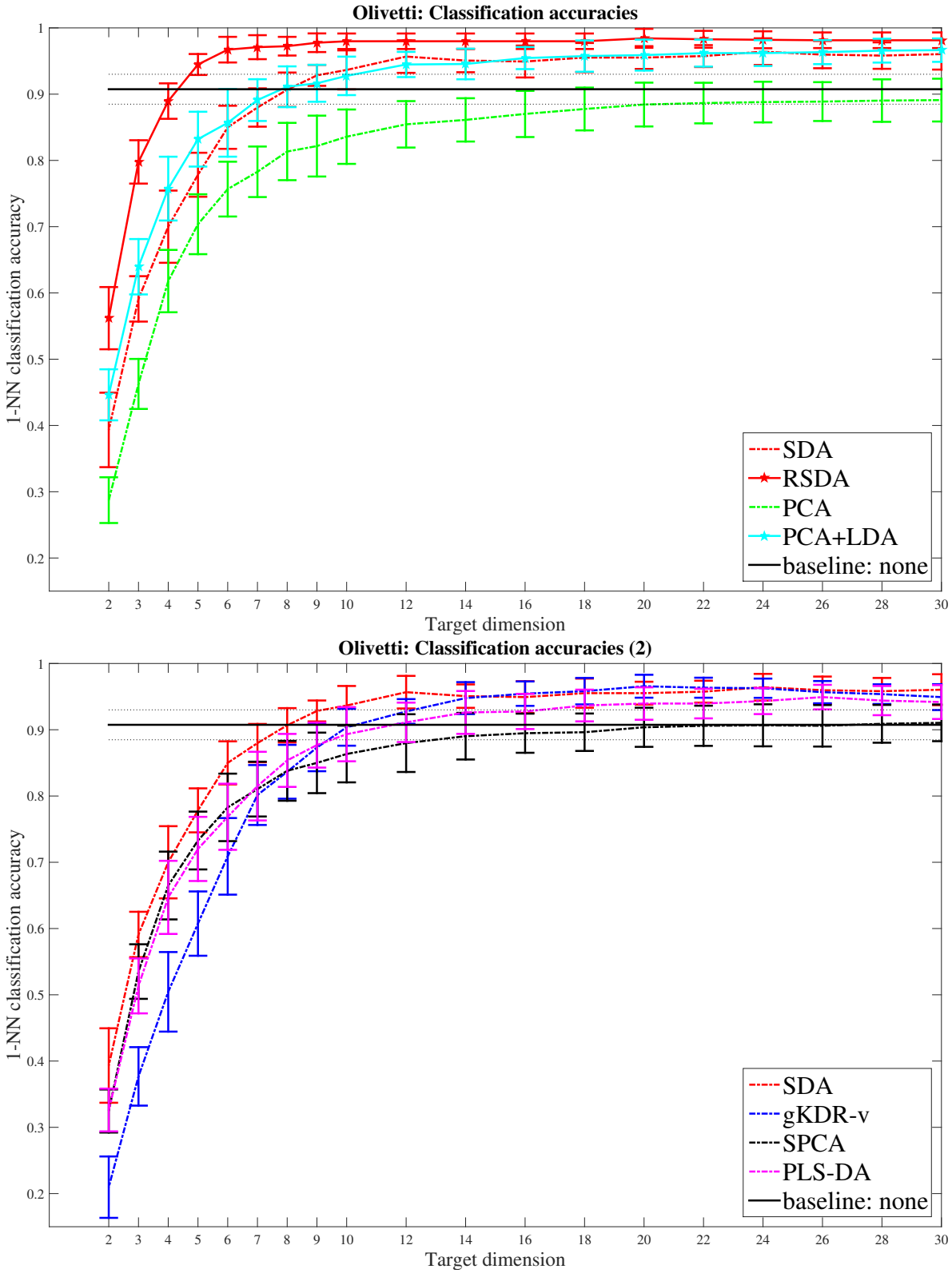

Figure 6.5: **Olivetti dataset.** Classification accuracies with a 1-$\mathcal{NN}$ classifier after projection with different dimension reduction methods. The baseline is the classification accuracy in the original high-dimensional dataset.

Figure 6.6: The five first orthogonal directions in a RSDA embedding of the Olivetti faces. White areas denote positive pixel weights, and black values denote negative pixel weights. Red values are neutral values.

Figure 6.7: The five first orthogonal directions in a LDA embedding of the Olivetti faces. White areas denote positive pixel weights, and black values denote negative pixel weights. Red values are neutral values.

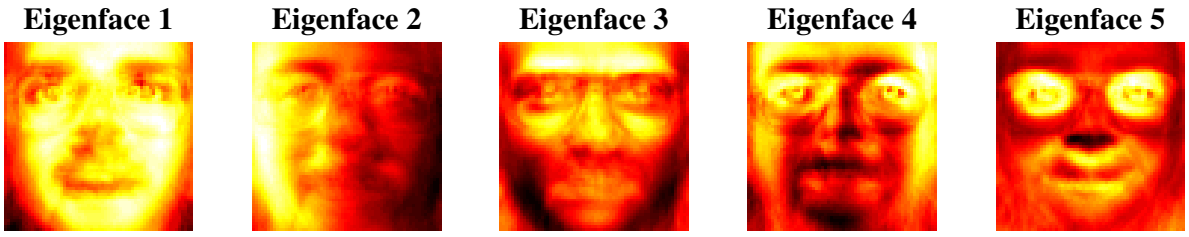

Figure 6.8: The five first orthogonal directions in a PCA embedding of the Olivetti faces. White areas denote positive pixel weights, and black values denote negative pixel weights. Red values are neutral values.

eyebrows. The RSDA embeddings put much weight on prototype faces 2, 3 and 4. The eyebrows, the nose and the eyes are important contributing factors in determining the projection location.

In general, the reprojected subjects in prototype or fisherfaces look similar, even if the subject is rotating his/her head or grimacing. In these supervised methods, the projections focus on some particular detail of the image that remains similar in all images. PCA does not contain any learning information and the reconstructed images can end up fairly far away from each other. This contributes to the larger standard error in Table 6.2.

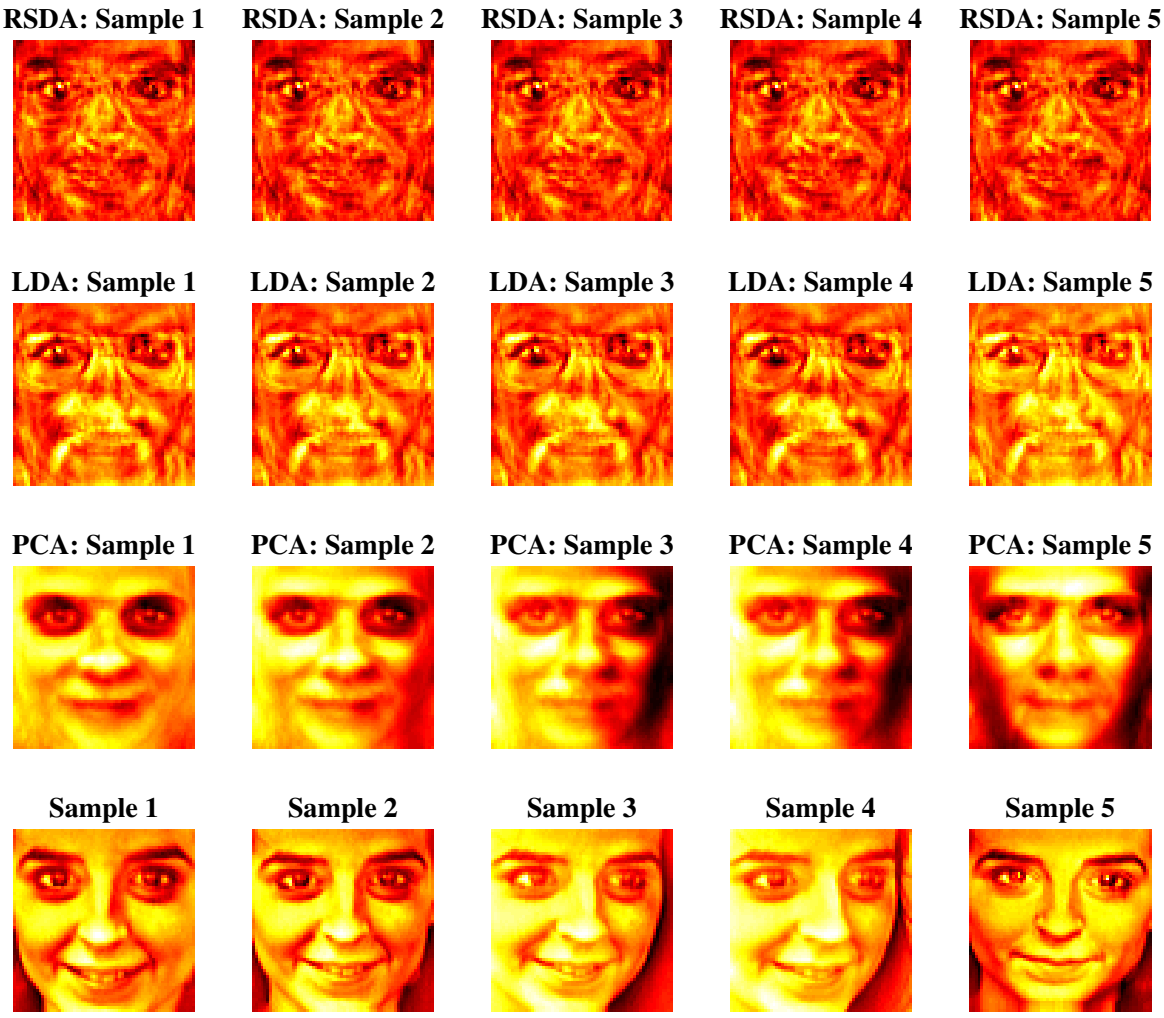

Figure 6.9: A comparison of a persons image and different reconstructions of the image.

Table 6.2: Proportions of prototype faces used to construct the low-dimensional embedding in Figure 6.9. Mean plus standard error.

Vector	RSDA	LDA	PCA
v_1	$-13.81\% \pm 1.21\%$	$34.48\% \pm 4.58\%$	$6.28\% \pm 23.22\%$
v_2	$34.44\% \pm 0.92\%$	$7.89\% \pm 2.29\%$	$25.19\% \pm 12.77\%$
v_3	$-24.82\% \pm 1.33\%$	$2.81\% \pm 6.82\%$	$16.96\% \pm 8.37\%$
v_4	$-21.23\% \pm 2.09\%$	$29.98\% \pm 2.74\%$	$-20.78\% \pm 3.27\%$
v_5	$5.70\% \pm 1.86\%$	$-21.88\% \pm 3.72\%$	$-19.39\% \pm 16.32\%$

Table 6.3: Mean classification accuracies on the USPS dataset with an 1-$\mathcal{NN}$ classifier a range of reduced spaces acquired with different DR methods. The best results are bold-faced.

Dim.	SDA	PCA	PLS-DA	SPCA	gKDR-v	LDA
$2D$	**67**	46	49	49	42	55
$4D$	**85**	69	77	76	57	78
$6D$	**90**	81	87	88	73	89
$8D$	**92**	87	**92**	**92**	83	**92**
$10D$	92	92	**93**	92	88	92
$14D$	92	**94**	**94**	92	93	82
$18D$	92	**95**	**95**	92	**95**	92
$22D$	93	**96**	95	92	**96**	92
$26D$	93	**96**	95	92	**96**	92
$30D$	93	**96**	95	92	**96**	92

6.3 The USPS dataset

The US Postal Service [55] dataset contains 9298 hand-written images of digits. Each digit is represented by 16-by-16 pixel grey-scale images. The data was divided randomly so that two thirds was used for training and one third was used for testing. The random selection was repeated ten times to obtain error bars.

The 1-$\mathcal{NN}$ classification accuracy of out-of-sample test points projected with some dimension reduction tools are shown in Figure 6.10. SDA has the highest accuracies in small dimension reduction tasks. We can observe a saturation in the classification accuracies of LDA, SPCA and SDA. The performance of these methods is similar to that of applying a linear SVM on the dataset, which had 11.3% error rate in the USPS dataset [42]. The discretization of the hand-written images is low-resolution, which makes it difficult to find meaningful low-dimensional projections of the data. The saturation is related to the fact that the defined optimal simplex structure of the data is reached already at 9 dimensions. PCA, PLS-DA and gKDR-v approach or exceed the initial classification accuracy 96.3% when using 30 dimensions or so. The mean classification accuracies are shown in Table 6.3 too.

Linear two-dimensional embeddings of the data points are compared in Figure 6.11. We can see that many methods resemble multidimensional simplexes projected onto a subspace with too many classes crowding near the origin. Such projections are not ideal in the presence of multiple classes. On the contrary, SDA manages to fill a two-dimensional circle, ultimately resulting in a higher class discrimination ability. We can see that the second vector in KSPCA is poorly scaled, resulting in poorer discrimination ability than SPCA. The SDPP structure is similar to the SDA structure. The data is sampled frequently, giving a better SDPP projection than in the Olivetti dataset.

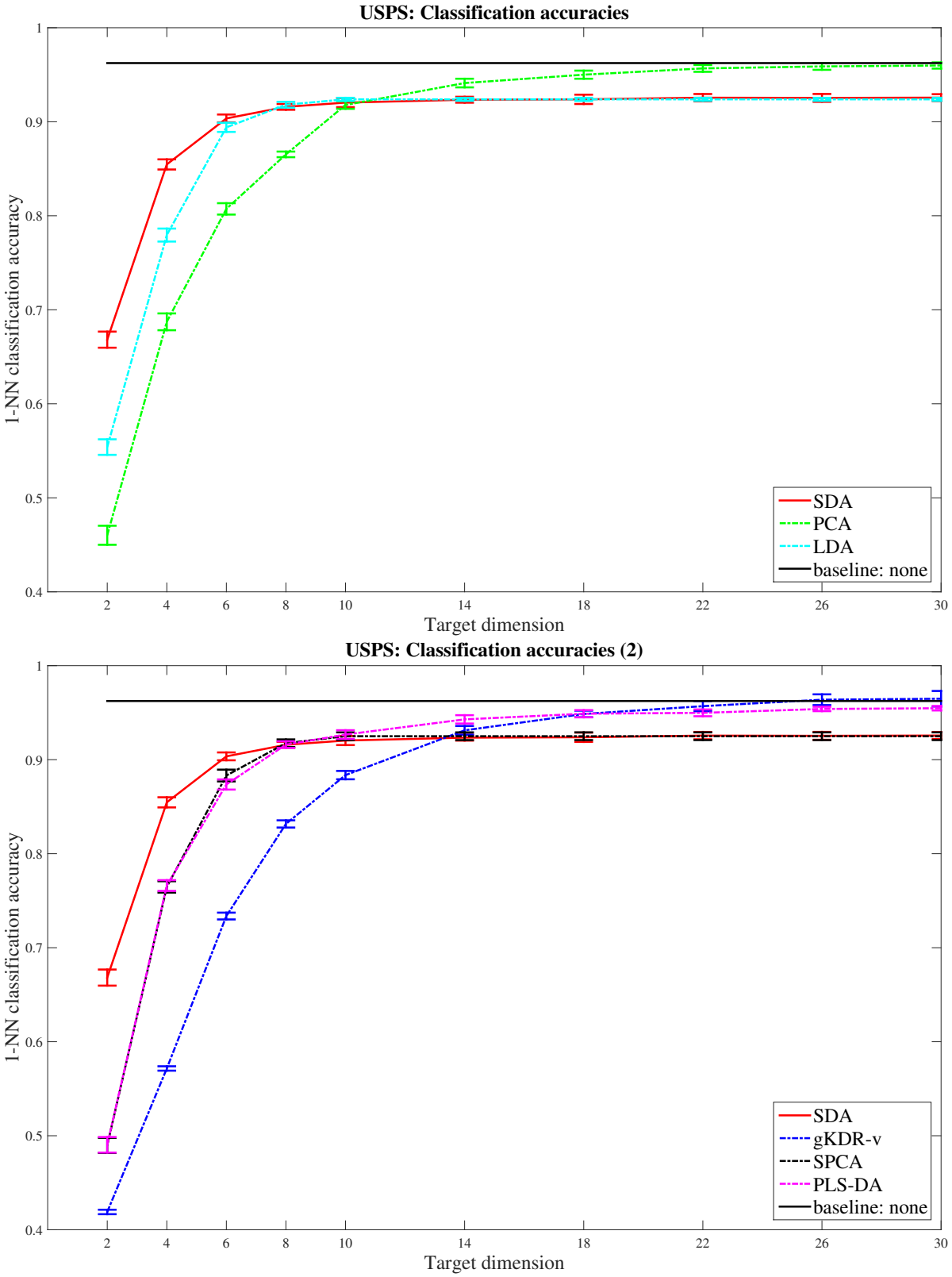

Figure 6.10: **USPS dataset.** Classification accuracies for different DR methods. The baseline is the classification accuracy in the original high-dimensional dataset.

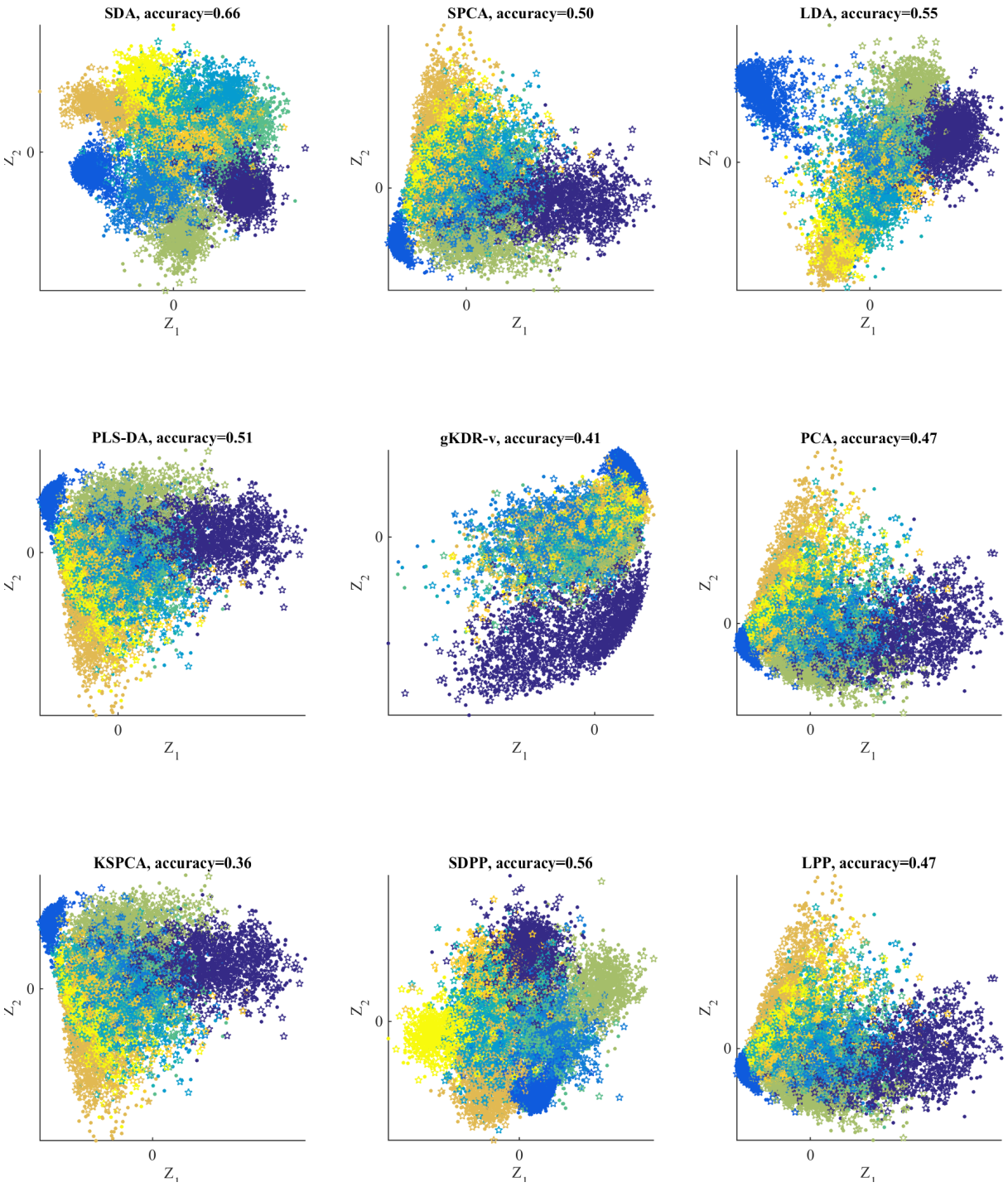

Figure 6.11: **Linear embeddings of the USPS dataset.** Dots denote projected learning points and stars denote projected test points. The 1-$\mathcal{NN}$ classification accuracy resulting from this embedding is added to the title.

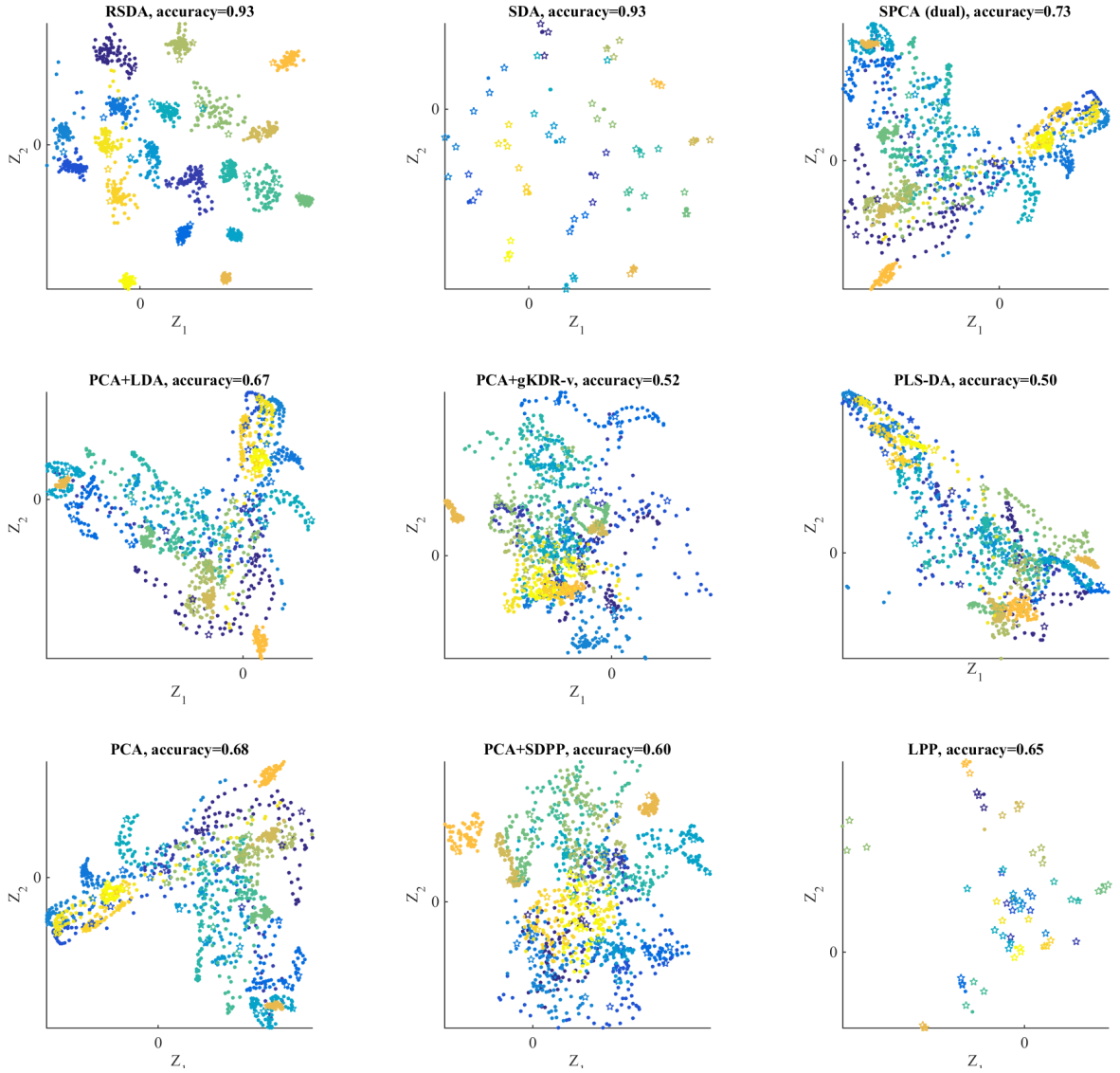

Figure 6.12: **Linear embeddings of the COIL-20 dataset.** Dots denote projected learning points and stars denote projected test points. The 1-$\mathcal{NN}$ classification accuracy resulting from this embedding is added to the title.

6.4 COIL-20 Object Images

The Columbia Object Image Library contains rotating images of 20 objects, photographed at 5 degree intervals [48]. The dataset was summarized in Chapter 5. Nine two-dimensional embeddings of the COIL-20 dataset are shown in Figure 6.12.

Figure 6.13 shows classification accuracies for the previous techniques calculated over the dimensions two to five. The means and standard errors were calculated by leaving three elements out of each class at each round and repeating the runs 24 times, thus going through the whole data. The tolerance for the SDA algorithms was set at 10^{-5}. SDA and RSDA can in average identify over 90% of the classes in with two variables. At five dimensions, most algorithms perform similarly. The mean classification accuracies are shown in Table 6.4.

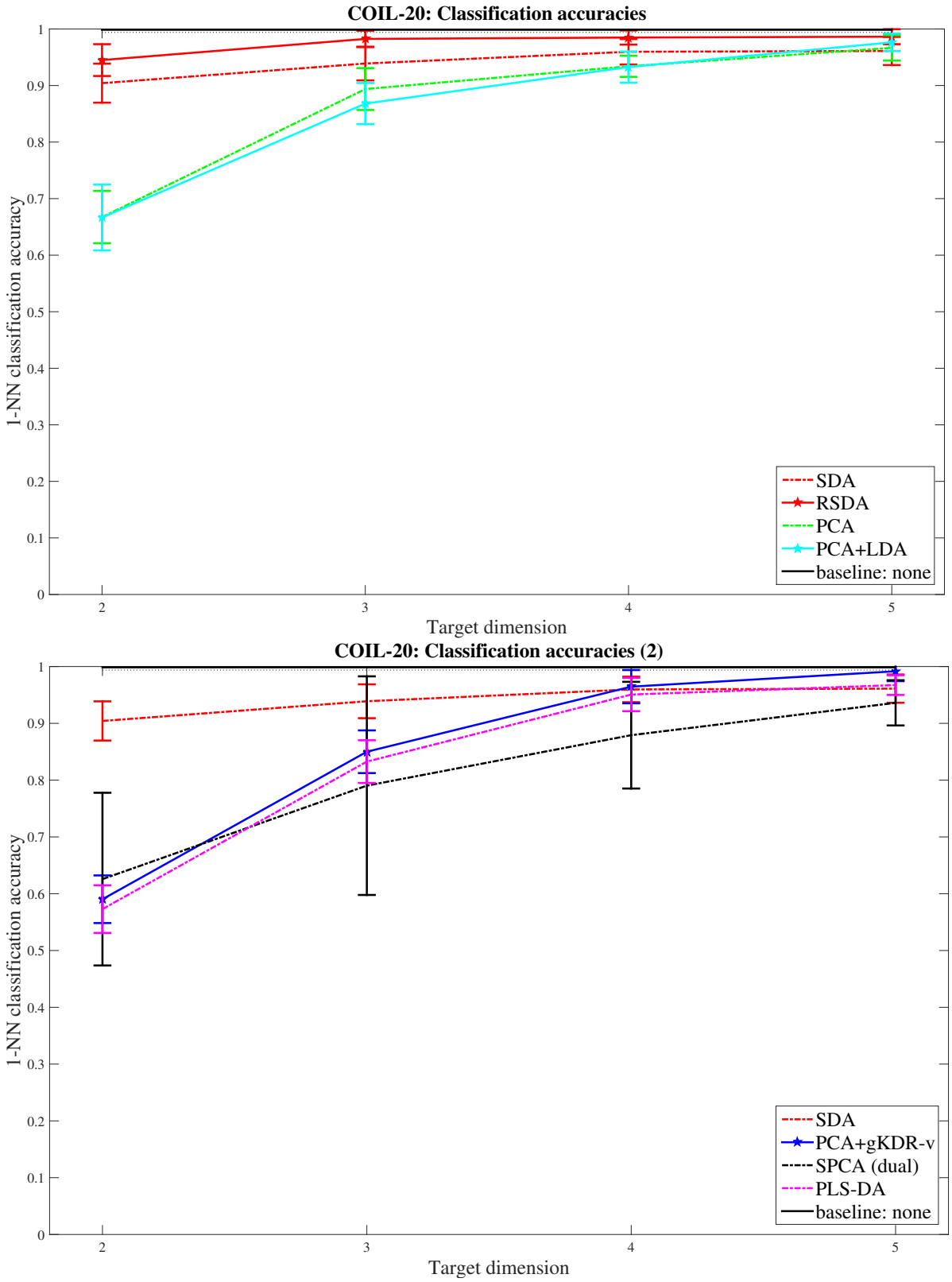

Figure 6.13: **COIL-20 dataset.** Classification accuracies for different DR methods. The baseline is the classification accuracy with no DR at all.

Table 6.4: Mean classification accuracies on the COIL-20 dataset with an 1-$\mathcal{NN}$ classifier a range of reduced spaces acquired with different DR methods.

Dim.	SDA	RSDA	PCA	PLS-DA	SPCA	gKDR-v	LDA
$1D$	**62**	57	25	27	25	20	26
$2D$	90	**95**	67	57	63	59	67
$3D$	94	**98**	89	83	79	85	87
$4D$	96	**98**	93	95	88	96	93
$5D$	96	**99**	97	97	94	**99**	98

6.5 Computational complexity and runtime comparison

The computational complexity of SDA in gradient based methods is largely determined by the number of times the gradient in Equation (4.6) is evaluated. The matrix expression has the time complexity $O(dn^2+dDn)$, where D is the dimensionality of the input space, d is dimensionality of target space and n is the number of samples. As such, optimizers that require as few function evaluations as possible would be efficient choices. If the Hessian information is used as well, the complexity is $O(D^3+D^2n+Dn^2+n^2d+nDd)$, including the gradient evaluation, the partial Hessian evaluation, the Cholesky factorization and the backsolves. The use of the partial Hessian might be efficient if the input dimension is not very high.

The processing time of the algorithms in Table 6.5 is compared on the three featured datasets in Figure 6.14. The fastest algorithm differs depending on the characteristics of the dataset. The spectral direction method converges faster and at a lower cost than the other algorithms in the USPS dataset. Convergence is reached in about two minutes. The number of variables is still small enough so that the partial Hessian information can be utilized cost-efficiently. The Olivetti and COIL-20 datasets contain a much larger number of variables. In COIL-20, the partial Hessian is re-evaluate only every 20 iterations to increase the performance. The LBFGS algorithm and different forms of the nonlinear conjugate gradient method are faster choices when performing dimension reduction for very high-dimensional spaces. The use of the spectral direction algorithm seems justified when the number of input dimensions is small enough. In general, we can see that the LBFGS is among the top algorithms in all datasets, making it a good default algorithm.

Table 6.5: Different compared optimization algorithms.

Acronym	Method
GD:	Gradient descent [56]
BB:	GD with Barzilai and Borwein step length [56]
CG:	Conjugate gradient (Hestenes-Stiefel update) [56]
PCG:	Preconditioned CG (LBFGS preconditioning)[56]
RCG:	Conjugate gradient (Polak-Ribiere update) [52]
LBFGS:	Limited-memory BFGS [56]
SD:	Spectral direction (Modified Newton's method) [56]

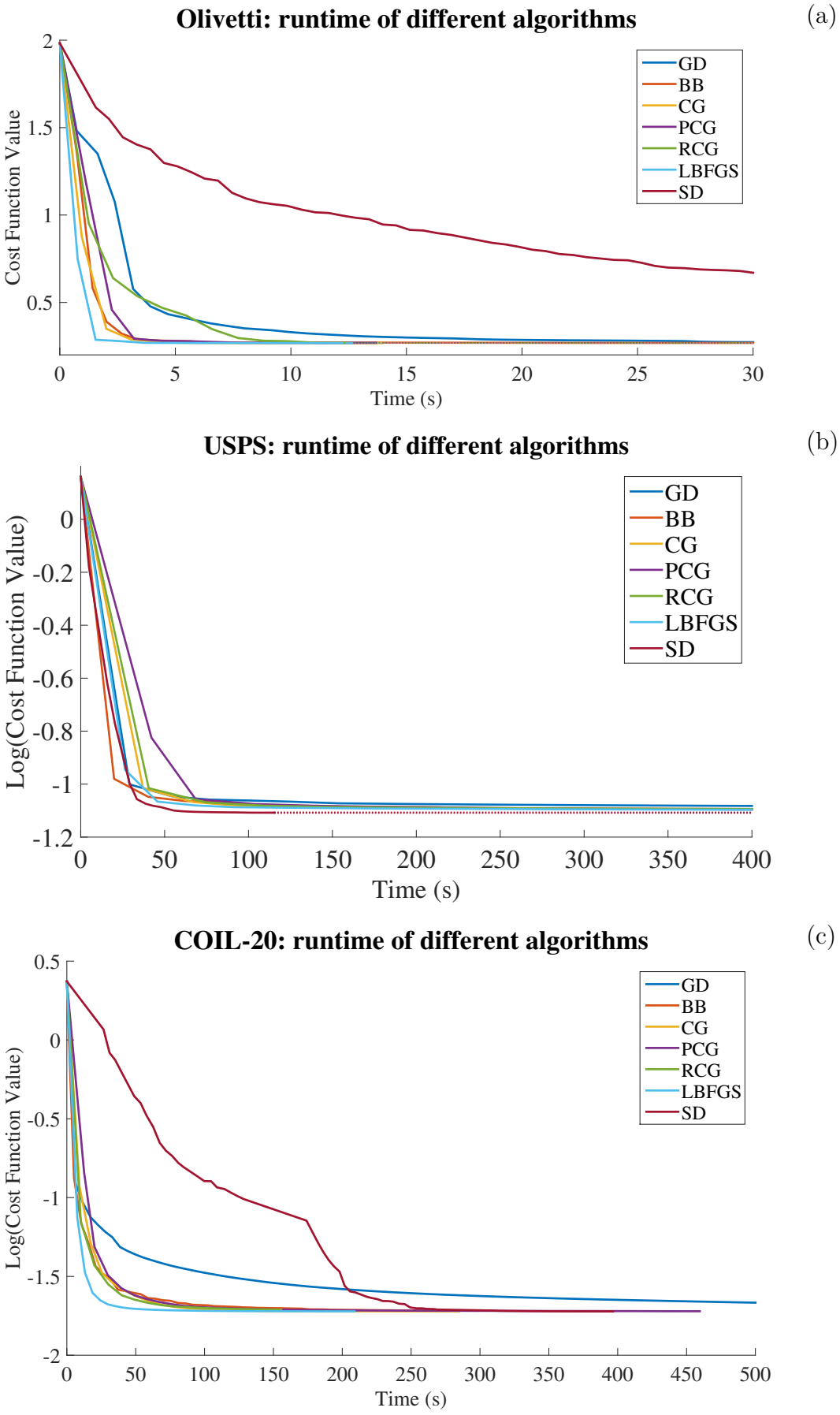

Figure 6.14: **Runtimes with different optimization algorithms.** The fastest methods differ depending on the characteristics of the datasets.

6.6 Comparisons on more datasets

In this subsection we compare the proposed method with state-of-the-art linear embeddings especially for visualization settings in two dimensions. The algorithms were run over three standard UCI datasets [5], three large datasets (more than 4000 data points) and three very high-dimensional datasets (more than 4000 input dimensions). In general, the algorithms were run for different selections of training and test points 10 times to obtain the confidence intervals. The COIL-20 and COIL-100 datasets were evaluated in the principle of leave-*three*-out, as discussed in subsection 6.4. In the tables that follow, a distinction is made between different dimension reduction types: *none*, *supervised* and *unsupervised*. The different types are separated by horizontal lines.

6.6.1 UCI datasets

The UCI datasets Iris, Wine and Wisconsin Breast Cancer displayed next are all standard datasets with rather few input dimensions. The resulting of two-dimensional projections are shown in Table 6.6. For these UCI datasets, all methods perform quite similarly. The tests were repeated 20 times to obtain the error bars.

Table 6.6: Three UCI datasets: 1-$\mathcal{NN}$ generalization accuracy ("mean ± std") on test set. The datasets were reduced to two dimensions, except for *None*, in which no dimension reduction was done.

Method	Iris	Wine	W. Breast Cancer
None	0.941 ± 0.026	0.949 ± 0.026	0.957 ± 0.014
SDA	0.948 ± 0.030	$\mathbf{0.983 \pm 0.017}$	0.956 ± 0.009
SDPP	0.920 ± 0.037	0.959 ± 0.035	0.940 ± 0.019
LDA	$\mathbf{0.962 \pm 0.025}$	0.981 ± 0.016	$\mathbf{0.961 \pm 0.009}$
PLS-DA	0.879 ± 0.040	0.974 ± 0.021	0.957 ± 0.008
gKDR	0.960 ± 0.021	0.959 ± 0.030	0.956 ± 0.013
SPCA	0.892 ± 0.026	0.974 ± 0.018	$\mathbf{0.961 \pm 0.011}$
KSPCA	0.893 ± 0.047	0.971 ± 0.019	0.893 ± 0.087
PCA	0.860 ± 0.034	0.938 ± 0.024	$\mathbf{0.961 \pm 0.011}$
LPP	0.889 ± 0.041	0.923 ± 0.017	0.953 ± 0.013

6.6.2 Large datasets

Three large datasets were compared. Two datasets were optical number recognition tasks (MNIST, USPS) and one was a phoneme recognition dataset. In SDA, the optimality tolerances for the large datasets were set at 10^{-5} and the tests were repeated 10 times each. The results can be seen in Table 6.7. SDA performs the best in all tests. For comparison, RSDA was also calculated. The performance only increased 0.2 percent units in average in the phoneme dataset.

[III] Dimension reduction done in a PCA reduced space, with the size 100.

Table 6.7: Three large high-dimensional datasets, 1-$\mathcal{NN}$ generalization accuracy ("mean ± std") on test set. The datasets were reduced to two dimensions, except for *None*, in which no dimension reduction was done.

Method	Phoneme	MNIST5k	USPS
None	0.889 ± 0.010	0.936 ± 0.002	0.962 ± 0.002
SDA	0.875 ± 0.009	$\mathbf{0.557 \pm 0.006}$	$\mathbf{0.668 \pm 0.009}$
RSDA	$\mathbf{0.877 \pm 0.009}$	0.550 ± 0.005	0.665 ± 0.008
LDA	0.664 ± 0.010	0.461 ± 0.011[III]	0.554 ± 0.008
PLS-DA	0.779 ± 0.014	0.301 ± 0.006	0.490 ± 0.008
gKDR-v	0.809 ± 0.015	0.323 ± 0.024	0.453 ± 0.009
SPCA	0.780 ± 0.008	0.401 ± 0.008	0.490 ± 0.008
KSPCA	0.781 ± 0.009	0.401 ± 0.009	0.354 ± 0.010
PCA	0.765 ± 0.007	0.383 ± 0.006	0.460 ± 0.010

6.6.3 High-dimensional datasets

A face recognition dataset (Olivetti faces) and two object recognition datasets (COIL-20 and COIL-100) were compared. The regularized version of SDA was also calculated. The 1-$\mathcal{NN}$ out-of-sample classification accuracies are shown in Table 6.8. The proposed regularized algorithm RSDA has the highest accuracy among the tested algorithms on all datasets. The tests were repeated 10 times to obtain the error bars. The tolerance for optimality was set at 10^{-5} in Olivetti and COIL-20 and at 10^{-4} in COIL-100. The tolerances for the regularization search were set at one magnitude higher than the final algorithm, at 10^{-4} resp. 10^{-3}. Optimizing with RSDA, including the λ search procedure, was in average faster than using no regularization ($\lambda = 0$) in COIL-100, with the median time 88 min vs. 215 min[IV].

The COIL-100 dataset is visualized in Figures 6.15 and 6.16. The methods were projected with the same subset of the training and test data vectors and the same initialization (PCA). The SDA embedding managed to draw almost every elements into their own cluster. Projecting new data points was more difficult: 30% of the test points were projected close to their own class. The large number of input dimensions allowed SDA to find this structure. The regularized method looks messier, but its generalization ability is better. The clusters are on approximately the same locations in both methods, a result of the initialization.

[IV] Run on 6-core Intel Xeon X5650 @ 2.67GHz. We acknowledge the computational resources provided by Aalto Science-IT project.

[II] Dual formulation for SPCA used.

[III] Dimension reduction done in a PCA reduced space, with the size 100.

Table 6.8: Three very high-dimensional datasets, 1-$\mathcal{NN}$ generalization accuracy ("mean ± std") on test set. The datasets were reduced to two dimensions, except for *None*, in which no dimension reduction was done.

Method	Olivetti faces	COIL-20	COIL-100
None	0.908 ± 0.023	0.999 ± 0.005	0.988 ± 0.006
SDA	0.393 ± 0.056	0.904 ± 0.035	0.277 ± 0.024
RSDA	$\mathbf{0.562 \pm 0.047}$	$\mathbf{0.944 \pm 0.026}$	$\mathbf{0.605 \pm 0.026}$
LDA[III]	$0.446 \pm 0.039^{\text{III}}$	$0.656 \pm 0.079^{\text{III}}$	$0.300 \pm 0.054^{\text{III}}$
PLS-DA	0.310 ± 0.042	0.573 ± 0.042	0.481 ± 0.049
gKDR-v	0.210 ± 0.046	$0.565 \pm 0.057^{\text{III}}$	$0.142 \pm 0.038^{\text{III}}$
SPCA	0.325 ± 0.033	$0.623 \pm 0.152^{\text{II}}$	$0.437 \pm 0.061^{\text{II}}$
KSPCA	0.322 ± 0.037	0.567 ± 0.191	0.397 ± 0.055
PCA	0.289 ± 0.029	0.667 ± 0.046	$0.288 + 0.036$

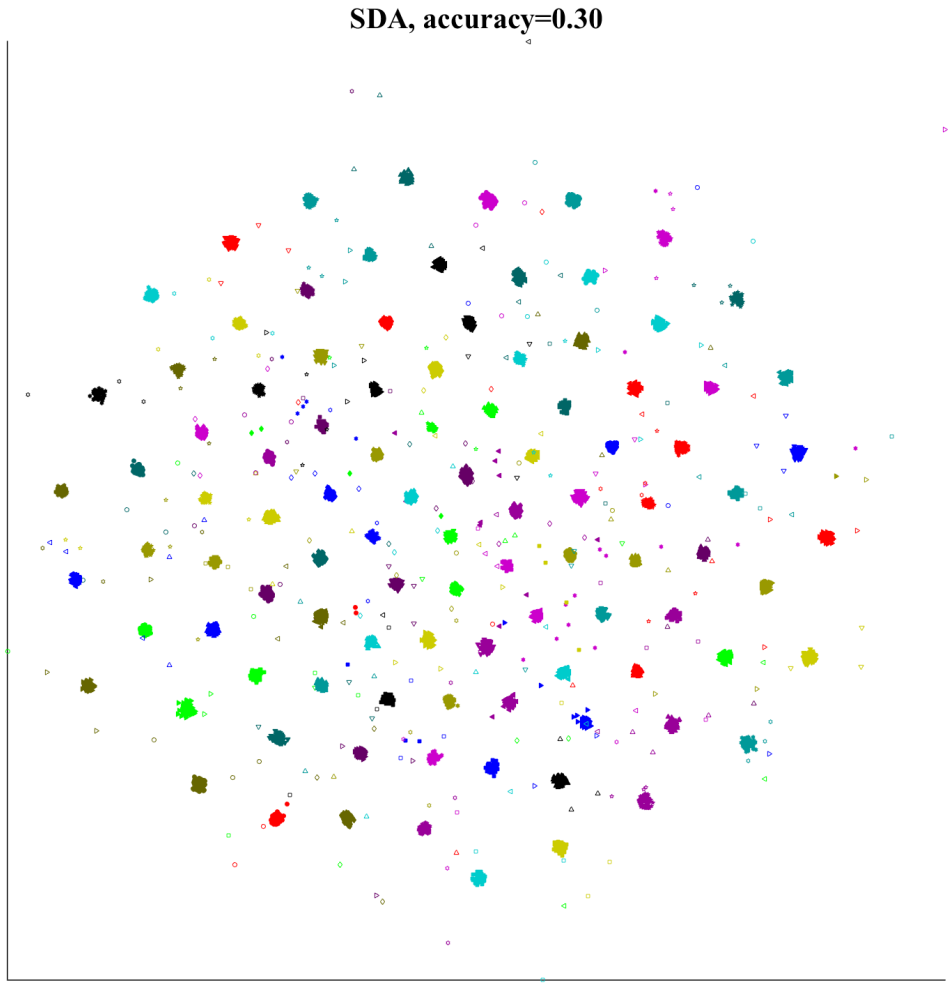

Figure 6.15: **A SDA linear embedding of the COIL-100 dataset.** Each class has a unique symbol and color combination. Filled markers denote learning points, hollow markers denote test points.

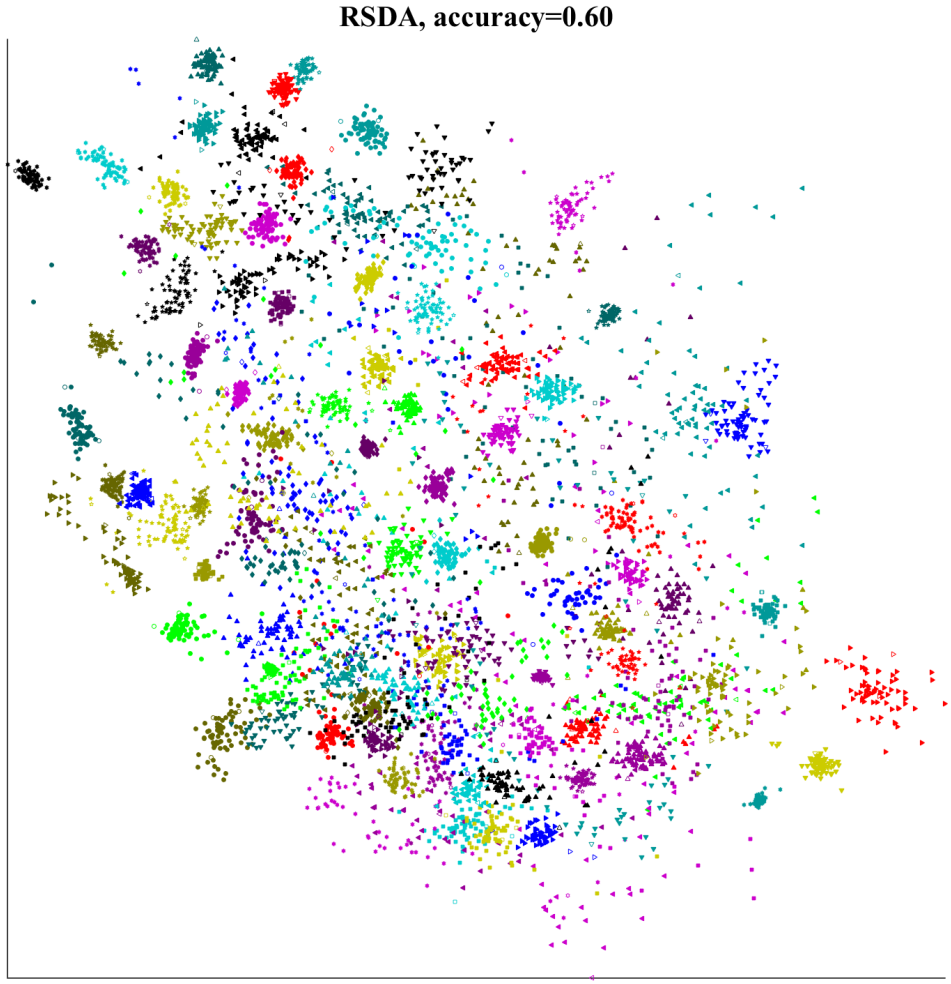

Figure 6.16: **A RSDA linear embedding of the COIL-100 dataset.** Each class has a unique symbol and color combination. Filled markers denote learning points, hollow markers denote test points.

Chapter 7

Conclusions

In this thesis, we have investigated a new dimensionality reduction method, Stochastic Discriminant Analysis. It has analogies with both the Supervised Distance Preserving Projections (SDPP) and Linear Discriminant Analysis (LDA) and works with categorical data in supervised classification problems. SDA and SDPP are connected through the Bregman divergences.

The method relies on a non-linear transformation of point-to-point distances of the low-dimensional projection space, obtained with a linear projection. The transformation matrix projecting the data from the high-dimensional space to the low-dimensional space is optimized by minimizing the Kullback-Leibler divergence, leading to a nonlinear unconstrained optimization problem. The projection maximizes class discrimination by minimizing the Kullback-Leibler divergence (M-projection) between a model distribution and a target distribution. Point-to-point distances in the low-dimensional space are evaluated through the heavy-tailed t-distribution with one degree of freedom. The cost function penalizes projecting points too close to points of another class, while at the same time it penalizes projecting points too far away from points of the same class. The distances between classes become constrained because of the zero-avoiding effect in the M-projection, which builds a tension in the projection: classes need to be separated but they cannot be too far away from other classes either.

The performance of SDA was compared against state-of-the-art supervised dimension reduction methods by the means of projecting out-of-sample test points and labeling these point with the 1-$\mathcal{NN}$ classifier. The performance is regularly better than state-of-the-art methods have in extremely low-dimensional embeddings. The classification accuracy increased until the dimension ν-1 was reached. The saturation is due to the defined optimal embedding, which is a simplex structure that can be found in ν-1 dimensions. The tests concluded that SDA works relatively better than other methods with very low-dimensional embeddings of multiclass data data points (the target dimensionality is much lower than the amount of classes), while the performance is similar to LDA and SPCA in higher dimensions.

The SDA method was regularized with Tikhonov regularization. Regularization increased the retrieval of meaningful low-dimensional embeddings in terms of out-of-sample classification accuracy when reducing the dimension of very high-dimensional datasets, such as images. For these datasets, the variables in the low-dimensional projection matrix can be reconstructed as images, and the projection vectors form faces in a similar manner as PCA or LDA projections form eigen- and fisherfaces.

The experimental section evaluated different optimization algorithms and found the LBFGS implementation to be an efficient choice in most tested cases. The LBFGS algorithm is efficient at solving large unconstrained optimization problems. The spectral gradient optimization algorithm, using the positive semidefinite partial Hessian, is efficient at solving problems involving a smaller amount of input dimensions, such as found in the USPS dataset.

Bibliography

[1] Ill-posed problems. Encyclopedia of Mathematics. Online, accessed 12.3.2015: http://www.encyclopediaofmath.org/index.php/Ill-posed_problems.

[2] MATLAB: Factorizations. Online, accessed 12.3.2015: http://se.mathworks.com/help/matlab/math/factorizations.html.

[3] Adragni, K. P., and Cook, R. D. Sufficient dimension reduction and prediction in regression. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 367*, 1906 (2009), 4385–4405.

[4] Alpaydin, E. *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[5] Bache, K., and Lichman, M. UCI machine learning repository, 2013. Online, accessed 12.3.2015: http://archive.ics.uci.edu/ml.

[6] Baker, K. Singular value decomposition tutorial. *The Ohio State University* (2005). Online, accessed 12.3.2015: http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf.

[7] Banerjee, A., Guo, X., and Wang, H. On the optimality of conditional expectation as a bregman predictor. *IEEE Trans. on Inf. Theory 51*, 7 (2005), 2664–2669.

[8] Barber, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012. Online, accessed 13.3.2015: http://www0.cs.ucl.ac.uk/staff/d.barber/brml/.

[9] Barshan, E., Ghodsi, A., Azimifar, Z., and Zolghadri Jahromi, M. Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds. *Pattern Recognition 44*, 7 (2011), 1357–1371.

[10] Barzilai, J., and Borwein, J. M. Two-point step size gradient methods. *IMA Journal of Numerical Analysis 8*, 1 (1988), 141–148.

[11] Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 7 (1997), 711–720.

[12] Belkin, M., and Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14* (2001), T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds., pp. 585–591.

[13] Bengio, Y., Paiement, J.-F., Vincent, P., Delalleau, O., Le Roux, N., and Ouimet, M. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Adv. in Neural Inf. Proc. Sys. 16* (2004), 177–184.

[14] Bertsekas, D. P. *Nonlinear programming*. Athena scientific Belmont, 1999.

[15] Bingham, E., and Mannila, H. Random projection in dimensionality reduction: applications to image and text data. In *Proc. of the 7th ACM SIGKDD Int. Conf. Knowledge discovery and data mining* (2001), ACM, pp. 245–250.

[16] Bishop, C. M., et al. *Pattern recognition and machine learning*, vol. 4. springer New York, 2006.

[17] Costa, S. I., Santos, S. A., and Strapasson, J. E. Fisher information distance: a geometrical reading. *Discrete Applied Mathematics* (2014).

[18] Cover, T. M., and Thomas, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[19] Donoho, D. L., et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture* (2000), 1–32. Online, accessed 12.3.2015, `http://statweb.stanford.edu/~donoho/Lectures/AMS2000/AMS2000.html`.

[20] Ekedahl, E., Hansander, E., and Lehto, E. Dimension reduction for the black-scholes equation. *Department of Information Technology, Uppsala University* (2007). Online, accessed 12.3.2015: `https://www.it.uu.se/edu/course/homepage/projektTDB/vt07/Presentationer/Projekt3/Dimension_Reduction_for_the_Black-Scholes_Equation.pdf`.

[21] Fisher, R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics 7*, 7 (1936), 179–188.

[22] Fujiwara, K., Sawada, H., and Kano, M. Input variable selection for pls modeling using nearest correlation spectral clustering. *Chemometrics and Intelligent Laboratory Systems 118* (2012), 109–119.

[23] Fukuda, K. Computing the delaunay complex and the voronoi diagram. what does it mean and how to do it with available software? Online, accessed 12.3.2015: `http://www.cs.mcgill.ca/~fukuda/soft/polyfaq/node31.html`.

[24] Fukuda, K. What is the delaunay triangulation in R^d? Online, accessed 12.3.2015: `http://www.cs.mcgill.ca/~fukuda/soft/polyfaq/node30.html`.

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

[26] Fukumizu, K., and Leng, C. Gradient-based kernel dimension reduction for supervised learning. *arXiv preprint arXiv:1109.0455* (2011).

[27] Guyon, I., and Elisseeff, A. An introduction to variable and feature selection. *The Journal of Machine Learning Research 3* (2003), 1157–1182.

[28] Hastie, T., Tibshirani, R., and Friedman, J. Datasets for the elements of statistical learning. Online, accessed 12.3.2015: `http://statweb.stanford.edu/~tibs/ElemStatLearn/data.html`.

[29] Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning*, 2 ed. No. 1. Springer, 2009. Online, accessed 12.3.2015: `http://statweb.stanford.edu/~tibs/ElemStatLearn/`.

[30] Haykin, S. S. *Modern filters*. Macmillan Coll Division, 1989.

[31] He, X., and Niyogi, P. Locality preserving projections. In *Neural information processing systems* (2004), vol. 16, pp. 153–160.

[32] Hinton, G. E., and Roweis, S. T. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems 15* (2002), pp. 833–840.

[33] Hotelling, H. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych. 24* (1933).

[34] Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. Extreme learning machine: theory and applications. *Neurocomputing 70*, 1 (2006), 489–501.

[35] Huberty, C. J., and Olejnik, S. *Applied MANOVA and discriminant analysis*, vol. 498. John Wiley & Sons, 2006.

[36] Imai, J., and Tan, K. S. Dimension reduction approach to simulating exotic options in a Meixner Lévy market. *IAENG International Journal of Applied Mathematics 39*, 4 (2009), 265–275.

[37] Jolliffe, I. *Principal component analysis*. Wiley Online Library, 2002.

[38] Jordan, M. I. University of california, berkley, statistical learning theory lecture slides: The kernel trick, 2004. Online, accessed 12.3.2015: `http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf`.

[39] Juha, K. Aalto university, Machine Learning and Neural Networks lecture slides 13, 2014. Online, accessed 14.3.2015: `https://noppa.aalto.fi/noppa/kurssi/t-61.5130/luennot/T-61_5130_lecture_13__black-and-white.pdf`.

[40] Juuti, M. SDPP toolbox. Online, accessed 12.3.2015: `http://users.ics.aalto.fi/~mjuuti/sdpp/`.

[41] Luenberger, D. G., and Ye, Y. *Linear and nonlinear programming*, vol. 116. Springer, 2008.

[42] Maji, S., and Malik, J. Fast and accurate digit classification. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-159* (2009).

[43] Martin, C. D., and Porter, M. A. The extraordinary svd. *The American Mathematical Monthly 119*, 10 (2012), 838–851.

[44] Miche, Y., van Heeswijk, M., Bas, P., Simula, O., and Lendasse, A. TROP-ELM: a double-regularized ELM using LARS and tikhonov regularization. *Neurocomputing 74*, 16 (2011), 2413–2421.

[45] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[46] Nayar, S. K., Nene, S. A., and Murase, H. Columbia object image library (COIL-100). *Department of Comp. Science, Columbia University, Tech. Rep. CUCS-006-96* (1996). Online, accessed 12.3.2015: `http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php`.

[47] Nazareth, L. A relationship between the bfgs and conjugate gradient algorithms and its implications for new algorithms. *SIAM Journal on Numerical Analysis 16*, 5 (1979), 794–800.

[48] Nene, S. A., Nayar, S. K., and Murase, H. Columbia Object Image Library (COIL-20). Tech. rep.

[49] Pearson, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine 2* (1901), 559–572.

[50] Pérez-Enciso, M., and Tenenhaus, M. Prediction of clinical outcome with microarray data: a partial least squares discriminant analysis (pls-da) approach. *Human Genetics 112*, 5-6 (2003), 581–592.

[51] Rao, C. R. Some combinatorial problems of arrays and applications to design of experiments. *A Survey of Combinatorial Theory 349* (1973), 359.

[52] Rasmussen, C. E. MATLAB function: Nonlinear conjugate gradient minimizer. Online, accessed 12.3.2015: `http://www.gatsby.ucl.ac.uk/~edward/code/minimize/`.

[53] Rasmussen, C. E., and Nickisch, H. Gaussian processes for machine learning (gpml) toolbox. *The Journal of Machine Learning Research 11* (2010), 3011–3015.

[54] Reid, M. Meet the bregman divergences. Online, accessed 12.3.2015: `http://mark.reid.name/blog/meet-the-bregman-divergences.html`.

[55] Roweis, S. Data for MATLAB hackers. Online, accessed 12.3.2015: `http://www.cs.nyu.edu/~roweis/data.html`.

[56] Schmidt, M. minfunc: unconstrained differentiable multivariate optimization in matlab. "Online, accessed 12.3.2015: `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`".

[57] Schölkopf, B., and Smola, A. J. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[58] Shewchuk, J. An introduction to the conjugate gradient method without the agonizing pain, 1994. Online, accessed 13.3.2015: `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`.

[59] Shlens, J. A tutorial on Principal Component Analysis. *arXiv preprint arXiv:1404.1100* (2014).

[60] Toh, K. C., Todd, M., and Tütüncü, R. SDPT3 - a MATLAB software package for semidefinite programming. *Optimization Methods and Software 11* (1998), 545–581.

[61] Van der Maaten, L., and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research 9*, 2579-2605 (2008), 85.

[62] Vandenberghe, L., and Boyd, S. Semidefinite programming. *SIAM review 38*, 1 (1996), 49–95.

[63] Vladymyrov, M., and Carreira-Perpinan, M. Partial-Hessian strategies for fast learning of nonlinear embeddings. *arXiv preprint arXiv:1206.4646* (2012).

[64] Yang, J., and yu Yang, J. Why can LDA be performed in PCA transformed space? *Pattern Recognition 36*, 2 (2003), 563 – 566.

[65] Yang, Z., Peltonen, J., and Kaski, S. Scalable optimization of neighbor embedding for visualization. In *Proc. of the 30th Int. Conf. on Machine Learning (ICML-13)* (2013), pp. 127–135.

[66] Zhu, Z., Similä, T., and Corona, F. Supervised distance preserving projections. *Neural Processing Letters 38*, 3 (2013), 445–463.