Christian Igel

A Short Course in Data Mining

Lecture Notes

Version 0.1.9

Department of Computer Science
University of Copenhagen

Preface

This text is supplementary material for the courses "Data Mining".

The goal is to revise and extend this document over time. Please do not hesitate to give me feedback (igel@diku.dk), in particular, please point out errors (even small ones) and sections or paragraphs that are not clear or should be extended.

These lecture notes build on existing textbooks and reviews, especially by Bishop [2006], Wu et al. [2008], and Fayyad et al. [1996].

Contents

1

Introduction

Data mining, also referred to as *knowledge discovery in data*, aims at extracting useful knowledge from databases. The volume of available data is steadily increasing, both in terms of the number of records in our database as well as the number of attributes. But this wealth of data is useless without techniques that transfer these data into relevant knowledge. Because of the sheer complexity of the data, these techniques must be automatic or at least semi-automatic.

Data mining has recently emerged as an interdisciplinary field in computer science and applied statistics. It is has become an essential tool in finance and industry, in the natural and health sciences, as well as in engineering. Data mining has an increasing impact on our society, opening new opportunities and bearing risks at the same time.

Example 1.1 (Searching the internet). A perfect example for data mining in our everyday life is using search engines to find web pages in the internet. This search is significantly different from a "simple" database query. We expect the search engine not only to find pages that match our search terms, but also to rank these according to their significance – in the sense that those pages most interesting for us should be ranked first. Clearly, the availability of powerful search engines has changed the way we are using the internet. This example underlines the need for efficient algorithms for mining large databases. Not only is the number of web pages huge, but usually also many attributes are needed for a proper description of the content of a page.

Example 1.2 (Verification of the nuclear test ban treaty). We have been dealing for instance with the analysis of hydroacoustic signals for verification of the Comprehensive Nuclear-Test-Ban Treaty (CTBT). The CTBT bans all nuclear explosions, for instance, nuclear weapon tests. The treaty is verified by the Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO). The CTBTO has established a monitoring network for detecting nuclear explosions consisting of 337 facilities located all over the

globe. The monitoring stations send their information to the CTBTO data center. These 15 GB of data per day have to be analyzed. This would be impossible without automatic filtering of the data.

1.1 The knowledge discovery process

Knowledge discovery and data mining starts with a database and with questions about these data. The first and arguably most important step when developing a data mining solution is to gain an understanding of these questions, the application domain, and available prior knowledge.

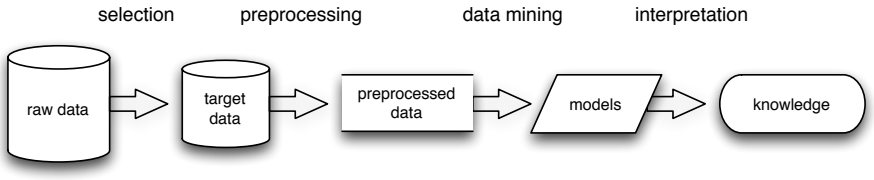

Fig. 1.1. Overview of the knowledge discovery in databases process, simplified and adapted from Fayyad et al. [1996].

Following the early review paper by Fayyad et al. [1996], the process of knowledge discovery in databases is schematically shown in Figure 1.1. We start from a database containing raw data. From these data, we select a proper subset relevant for our task at hand. This data is then preprocessed, that is, the data is transformed to suite the subsequent steps of processing. This also involves cleaning the data, for example, removing duplicate entries or wrong entries if these can be easily detected. Based on domain knowledge, missing data may be filled in (a process also referred to as data imputation). The data is then ready for the application of data mining algorithms as discussed in this mini-lecture. There is, however, no sharp distinction between the steps in Figure 1.1 (e.g., PCA as introduced in Chapter 5 can be viewed as both a preprocessing as well as a data mining technique).

1.2 Data mining tasks

Data mining can be applied to solve different kinds of tasks, the most important ones being:

Summarization: The aim of summarization is finding a compact/human-understandable description for (a subset of) the data.

Clustering: Clustering or segmentation means automatic grouping of data. Records in the same group should be more similar to each other than records from different groups. The notion of similarity is task specific.

Classification and regression: Classification is learning to map data records correctly to a finite number of classes. The learning is based on available data for which the right classes are known. The goal is to apply the classification rule to data for which the correct class is not known, in particular to future observations. Regression is similar to classification, except that the mapping sought-after should predict real-valued quantities instead of discrete classes.

Dependency modeling: In dependency modeling, we search for structural and quantitative relationships between attributes and model these mutual dependencies.

Probability density estimation: In density estimation, we infer from data the joint multivariate probability distribution of (a selection of) the attributes.

Change detection: This task, also called deviation, outlier, or novelty detection, aims at finding significant differences in the data compared to previous or normative values.

There is no crisp distinction between these tasks. A good estimate of the joint probability distribution of all attributes can be used to infer and quantify dependencies between attributes, the result of clustering can be a good way to summarize data, etc.

Example 1.3 (Stock market prediction). If we want to predict whether a particular share price goes up or down based on previous stock market values then we are dealing with a classification task. If we want to predict the actual share price, we have to solve a regression problem.

Example 1.4 (Market basket analysis). In market basket analysis of consumer data, the goal is to find rules describing which products – which *itemsets* – are frequently bought together. This information can be used for marketing purposes. This is a typical dependency modeling task.

Example 1.5 (Fraud detection). A prominent example of a change detection application is fraud identification. For instance, bank accounts can be monitored by a data mining algorithm that reports any significant deviation from an account's typical behavior because it may indicate some fraud.

1.3 Data mining as model building

Data mining can be understood as (semi-) automatically deriving a mathematical *model*, that is, a description of some phenomena using mathematical language, from data. Different kind of models can be distinguished, especially:

Descriptive models: Purely descriptive models convert the available raw data into a new – usually much more compact – representation that is better suited for subsequent processing. In particular, data mining techniques try to bring data into a human-understandable form.

Predictive models: Predictive models are derived from the available data to make prediction beyond the information stored in the database. Most important is the prediction of future behavior of some entities, but prediction also includes filling in unobserved data or generating data with the same properties as the observed records.

Classification and regression require predictive models. Predictive models can be further separated into two broad classes:

Generative models: Generative models try to approximate the process that has generated the available data. They can – in contrast to descriptive models – be used to create new database entries.

Conditional models: Conditional models try to make predictions about unobserved entities given observations.

The distinction between the different models is not sharp. Generative models, for example, may be a good description of the data. As we will see, they can also be used for conditional modeling. Still, it is often helpful to use these categorization when addressing a data mining task.

1.4 Outline of the course

From the broad field of data mining, only a subset of algorithms can be presented in this mini-lecture. We take a machine learning perspective. For this course, the algorithms have been selected based on the following criteria. All methods are basic, well established techniques and comparatively easy to understand. They are also straight-forward to implement and to apply. In particular, they do not have many hyperparameters that have to be adjusted (i.e., parameters of the algorithm itself that depend on the problem and have to be chosen in a proper way to ensure good performance). Understanding the presented methods can be viewed as a prerequisite for studying more sophisticated algorithms. The presented techniques are successfully used in practice and their simplicity does not imply that they do not perform well. Actually, I suggest to always apply one of these methods first, before trying more complex algorithms. The focus is on machine learning methods dealing with real-valued data.

1.5 Basic notation

In this mini-lecture, we will look at a restricted, still pretty general data mining scenario.

1.5.1 Data

Let the records in our database have the domain $\mathcal{X}$ and be denoted by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell \in \mathcal{X}$. Each record $\boldsymbol{x}_i$ has n attributes $x_{i1}, \ldots, x_{in}$. Records could correspond to websites, where x_{ij} measures how often a particular word indexed by j occurs in the website indexed by i. This *bag of words* representation is indeed frequently used when data mining is applied to text documents, an application domain referred to as *text mining*. In a bioinformatics scenario, records may correspond to gene sequences of a fixed length, where x_{ij} encodes the nucleotide at position j in sequence i. Or the $\boldsymbol{x}_i$ could represent 3D images, where every x_{ij} encodes a particular voxel of the image indexed by i.

In this course, we will mostly focus on real-valued attributes. That is, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{in})^{\mathrm{T}} \in \mathbb{R}^n = \mathcal{X}$. From the machine learning perspective, we often refer to elements of $\mathcal{X}$ as data points, to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ as the training data, and to $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$ as the training data set. Sometimes – in particular in data mining where the $\boldsymbol{x}_i$ are records in a database – it is more appropriate to view S not as a set but as a multi-set, in which the same object can occur more than once, or even a sequence, that is, an ordered multi-set. For traditional reasons, we nonetheless refer to S as the training set.

We gather all elements in S in the $\ell \times n$ *data matrix*

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{\ell 1} & x_{\ell 2} & \ldots & x_{\ell n} \end{pmatrix} , \tag{1.1}$$

in which the rows correspond to the records and the columns to the attributes. Note that in this common notation $\boldsymbol{x}_i$ is a column vector although it corresponds to a row in $\boldsymbol{X}$.

For predictive models, it makes sense to split the domain of the attributes into $\mathcal{X}$ and $\mathcal{Y}$ and consider data pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}$. Here $\boldsymbol{y}_i \in \mathcal{Y}$ denotes the quantity we want to predict given $\boldsymbol{x}_i \in \mathcal{X}$. The $\boldsymbol{y}_i$ are also called labels, targets, or dependent variables. For example, in classification $\mathcal{Y}$ is a discrete set of classes and in regression $\mathcal{Y} \subseteq \mathbb{R}^d, d \geq 1$. Note that the split of the database entries into $\mathcal{X}$ and $\mathcal{Y}$ is task-dependent. It is done for convenience to make the notation easier to understand.

Inferring a predictive model from training data $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_\ell, \boldsymbol{y}_\ell)\}$ is called *supervised learning*. The $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ can be viewed as examples given by a supervisor or teacher describing the mapping $\mathcal{X} \to \mathcal{Y}$. Because the models map from $\mathcal{X}$ to $\mathcal{Y}$, the $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are also called input patterns (inputs) and output patterns (outputs), respectively.

In clustering, there is no notion of a target or a given class label. Therefore clustering is often referred to a *unsupervised learning*.

1.5.2 Probabilistic point of view

We view the database entries $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ as realizations of a random vector X taking values in $\mathcal{X}$ (see Appendix A.1). The distribution of X is usually unknown. In other words, the entries in the database are a sample of the quantity we are interested in. For example, we may be interested in the general behavior of customers. But of course, our database just contains data from a few customers. Thus X describes the population of customers, and our $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ are samples from this population. This probabilistic point of view allows us to deal with the stochasticity and uncertainty in the data. If not stated otherwise, it is assumed that all data are identically, independently distributed (i.i.d.), that is, each pattern is an independent realization of a random variable with a stationary distribution p over $\mathcal{X}$.[1]

A typical task of generative modeling is to infer a model of p. In the case of conditional models, we assume that p can be factorized according to $p(X = x, Y = y) = p(X = x) \cdot p(Y = y|X = x)$, where $p(X = x)$ and $p(Y = y|X = x)$ are distributions over $\mathcal{X}$ and $\mathcal{Y}$, respectively (see Appendix A.1 if you are not familiar with the concept of a conditional probability as described by $p(Y = y|X = x)$). A generative model of $p(X = x, Y = y)$ leads to a conditional model $p(Y = y \,|\, X = x)$.

1.5.3 Objective functions

An important step in data mining – and machine learning in general – is to formalize the goal. We want to find a "good", an "appropriate" model. In order to do so, we have to quantify the quality of a model. This can be done by an *objective function*, which is often denote by J. It assigns a scalar quality to a model. The objective function often corresponds to an "error function", "risk", or "distortion measure". Then the smaller the objective function the better the model.

[1] In the following, the same symbol p is used for both probability density functions as well as probability mass functions. These specify continuous and discrete distributions of random variables, respectively, and therefore a distribution is often directly identified by p and with a standard abuse of notation the symbol p is also used for the underlying probability measures.

2

Nearest Neighbor Classification

The nearest neighbor classifier is a simple, still powerful classification algorithm.

2.1 Description of the algorithm

Assume that we are given a data set $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ and that we want to predict the label y of an unseen data point x.

The idea of nearest neighbor classification is to look in $x_1, \ldots, x_\ell$ for those k patterns that look most similar to x and to choose y based on their labels.

Algorithm 1: 1-nearest neighbour

Input: metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, training data $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$, new input x to be classified

Output: predicted label y of x

`/* ties are broken at random */`

1 $(x_{\min}, y_{\min}) = \operatorname{argmin}_{(x_i, y_i) \in S} d(x_i, x)$

Result: $y = y_{\min}$

In the simplest case, we just look for the pattern $x_{\min}$ that is most similar to x and assign x to the label of $x_{\min}$ (i.e., $y = y_{\min}$). This method is called 1-nearest neighbor classification (1-NN) and is shown in Algorithm 1.

In k-nearest neighbour classification (k-NN) shown in Algorithm 2, we look at the k nearest points and make a majority-vote. The label of x will be the label most frequent among the k selected neighbors.

To measure similarity and dissimilarity, we need the notion of a *metric* or *distance function*. A metric on $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with

1. $d(x, y) \geq 0$ with $d(x, y) = 0$ if and only if $x = y$,

Algorithm 2: k-nearest neighbour

Input: metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, training data $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$, new input x to be classified, number of neighbors k

Output: predicted label y of x

1 $S^\star \leftarrow \emptyset$

2 **while** $|S^\star| < k$ **do**

`/* ties are broken at random */`

3 $(x_{\min}, y_{\min}) = \operatorname{argmin}_{(x_i,y_i)\in S} d(x_i, x)$

4 $S \leftarrow S \setminus \{(x_{\min}, y_{\min})\}$

5 $S^\star \leftarrow S^\star \cup \{(x_{\min}, y_{\min})\}$

`/* ties are broken at random */`

Result: $y = \operatorname{argmax}_c |\{(x, y) \,|\, (x, y) \in S^\star \wedge y = c\}|$

2. $d(x, y) = d(y, x)$ (symmetry),
3. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

for all $x, y \in \mathcal{X}$.

The first property says that there are no negative distances and that if two objects have zero distance then they are the same. The second property simply states that the way from x to y is the same as from y to x. Finally, the last property states that the distance between two points is the shortest distance along any path.

Example 2.1. The standard metric in the $\mathbb{R}^n$ is the Euclidean metric given by

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

for $\boldsymbol{p} = (p_1, \ldots, p_n)^{\mathrm{T}}, \boldsymbol{q} = (q_1, \ldots, q_n)^{\mathrm{T}} \in \mathbb{R}^n$.

2.2 Training error and generalization

The 1-nearest neighbor classifier always achieves zero classification error on the training data S. Clearly, this does not guarantee zero error on unseen data.

Insight 2.1. The error on the data set used for building a model (the training data) is a bad predictor of its generalization performance. The generalization performance can be estimated by hold-out data not used in any stage of the model building process.

Zero error on the training data is not necessarily a desirable property, for instance for problems that are noisy in the sense that a data point may have different class labels each with a certain probability. We will come back to this issue in Chapter 3.

2.3 Cross-validation

The question arises how to choose the number of neighbors k. This parameter of the algorithm has to be specified *a priori*. Such parameters are often called *hyperparameters*. In general, hyperparameters influence what kind of models the algorithm should consider or prefer. Therefore their choice is often termed *model selection*.

Cross-validation (CV) is a standard technique for adjusting hyperparameters of predictive models. In κ-fold CV, the available data S is partitioned into κ subsets $S_1, \ldots, S_\kappa$. Each data point in S is randomly assigned to one of the subsets such that these are of almost equal size (i.e., $\lfloor |S|/\kappa \rfloor \leq |S_i| \leq \lceil |S|/\kappa \rceil$). Further, we define $S_{\setminus i} = \bigcup_{j=1,\ldots,\kappa \wedge j \neq i} S_j$ as the union of all data points except those in S_i. For each $i = 1, \ldots, \kappa$, an individual model is built by applying the algorithm to the training data $S_{\setminus i}$. This model is then evaluated using the test data in S_i. The average of the κ outcomes of the model evaluations is called *cross-validation (test) performance* or *cross-validation (test) error* and is used as a predictor of the performance of the algorithm when applied to S. Typical values for κ are 5 and 10 [Hastie et al., 2001].

To choose k for k-NN using κ-fold CV, we once split the S into κ subsets. Then we compute the CV error using this split error for k-NN classifiers using different values for k (say, $k = 1, 3, 5, \ldots$). Finally, we pick the k with the lowest CV error and use it for k-NN based on the complete data set S.

It must be noted that the data used for model selection must be independent from data for assessing the final performance of a model.

2.4 Exercises

Exercise 2.1. Consider a noisy binary classification problem in which the "true" label of a point (given by a deterministic function) is always changed by noise in 10 % of the observations (i.e., each data point belongs to one class with a probability of 90 %, and with 10 % to the other). What training error would you expect from an optimal classifier?

Exercise 2.2. Generate real-valued random vectors of dimensionality n, where each component is drawn independently from the same distribution with known finite mean and variance (e.g., a normal distribution). Compute the length (i.e., the Euclidean norm) of the vectors and the Euclidean distances from each other for increasing n. What do you observe? What happens for $n \to \infty$? What does this imply for nearest neighbor classification?

3

Linear Regression

The general goal of regression is to assign data records $\boldsymbol{x} \in \mathcal{X}$ to real-valued quantities $\boldsymbol{y} \in \mathbb{R}^m$. For some data records the corresponding quantities are known. These records are used to infer a general function for predicting the correct value of the target variable $\boldsymbol{y}$ for a given $\boldsymbol{x}$.

3.1 Description of the algorithm

Let us consider $\mathcal{X} = \mathbb{R}^n$ and $m = 1$. We learn a predictive model based on sample data $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell)\}$. In linear regression, we consider affine linear models of the form

$$y = f(\boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i + b = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b \tag{3.1}$$

with model parameters $\boldsymbol{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. That is, we assume a linear dependency between the input and the target. In regression, the target is often called dependent variable, regressand, or response variable. The components of the random variable X (see Section 1.5.2) are referred to as regressors, explanatory variables, input variables, or predictor variables. The parameter b, which is called bias, offset, or intercept, defines the value of the model if the input is $\mathbf{0}$. The regression coefficients gathered in $\boldsymbol{w}$ determine the effects of the regressors on the dependent variable. If $w_i = 0$, the ith input variable has no effect on the response variable in the model. If $w_i > 0$, the regressand increases with an increase of the ith input variable and decreases if the ith input decreases. Accordingly, if $w_i < 0$, the regressand decreases with an increase of the ith input variable and decreases if the ith input increases. The strength of these effects is determined by the amount $|w_i|$ of the regression coefficient. In the assumed linear model, the input variables influence the response variable independent of each other. This is a strong assumption.

For convenience, we define $\tilde{\boldsymbol{x}}_i^{\mathrm{T}} = (x_{i1}, \ldots, x_{in}, 1)$ for $i = 1, \ldots, \ell$ and $\tilde{\boldsymbol{w}}^{\mathrm{T}} = (w_1, \ldots, w_n, b)$ and consider the equivalent formulation

$$y = f(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{n+1} \tilde{w}_i \tilde{x}_i = \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}} \ . \tag{3.2}$$

The corresponding data matrix is given by

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} & 1 \\ x_{21} & x_{22} & \dots & x_{2n} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{\ell 1} & x_{\ell 2} & \dots & x_{\ell n} & 1 \end{pmatrix} \ . \tag{3.3}$$

To keep the notation uncluttered, we omit the tilde in the following although all considerations refer to the formulation (3.2).

The formal goal of linear regression is to minimize the *sum-of-squares error*

$$J = \sum_{i=1}^{\ell} \{y_i - f(x_i)\}^2 \ = \sum_{i=1}^{\ell} \left\{y_i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i\right\}^2 \ . \tag{3.4}$$

The quantity J/ℓ is called *mean-squared error* (MSE). The optimal parameters $\boldsymbol{w}^\star$ minimizing J are given by

$$\boldsymbol{w}^\star = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \ , \tag{3.5}$$

where $\boldsymbol{y} = (y_1, \dots, y_\ell)^{\mathrm{T}}$, see Algorithm 3.

Algorithm 3: linear regression

Input: $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_\ell, y_\ell)\} \in (\mathbb{R}^n \times \mathbb{R})^\ell$
Output: affine linear model

1 $\boldsymbol{y} = (y_1, \dots, y_\ell)^{\mathrm{T}}$

2 $\boldsymbol{X} = \begin{pmatrix} x_{11} & \dots & x_{1n} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{\ell 1} & \dots & x_{\ell n} & 1 \end{pmatrix}$

3 $(w_1, \dots, w_n, b)^{\mathrm{T}} = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}$

Result: model $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b$

Figures 3.1 and 3.2 show examples where the assumption of an affine linear model is appropriate and where it is not.

3.2 Deriving linear regression

In the following, we first prove that the parameters given by (3.5) are indeed optimal. Then we justify the sum-of-squares error measure (3.4).

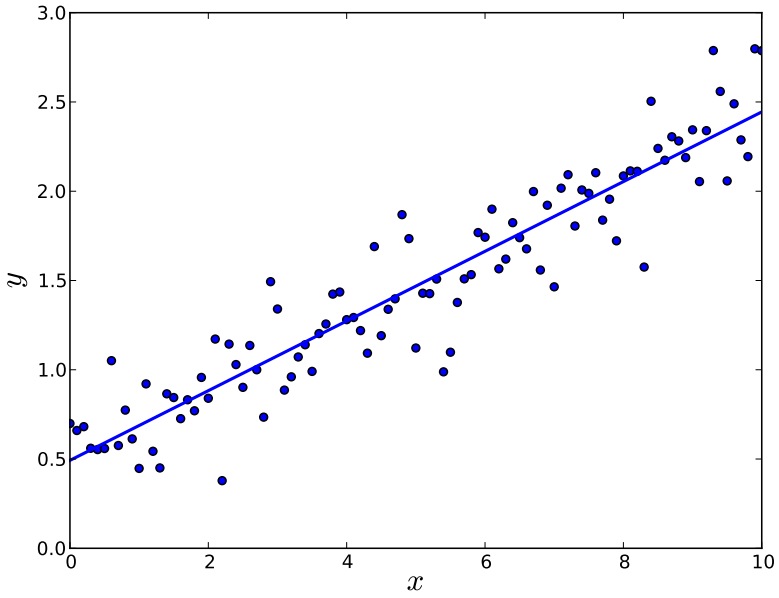

Fig. 3.1. Linear regression discovering a linear function that has been perturbed by noise.

3.2.1 Solution using pseudo-inverse

At a minimum of the objective function all partial derivatives (see section A.2.3) must be zero. The (for convenience halved) sum-of-squares error of the linear model has the partial derivatives

$$\frac{\partial}{\partial w_i}\frac{1}{2}J = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}^2 = -\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}x_i \tag{3.6}$$

for every component i of the parameter vector $\boldsymbol{w}$. This follows from

$$\begin{aligned}
\frac{\partial}{\partial w_i}\frac{1}{2}\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}^2 &\overset{\text{constant factor rule}}{=} \frac{1}{2}\frac{\partial}{\partial w_i}\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}^2 \\
&\overset{\text{power rule}}{=} \sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}\frac{\partial}{\partial w_i}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\} \\
&\overset{\text{constant factor rule}}{=} -\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}x_i \ ,
\end{aligned} \tag{3.7}$$

see Appendix A.2.2. The only reason for considering $\frac{1}{2}J$ instead of J is that for $\frac{1}{2}J$ all constants in the derivative disappear after applying the power rule. Setting the derivatives to zero gives

$$0 = \sum_{i=1}^{\ell} y_i x_i - \boldsymbol{w}^{\mathrm{T}} \sum_{i=1}^{\ell} \boldsymbol{x}_i x_i \tag{3.8}$$

for all i. These conditions can be combined in one equation using vector notation

$$\nabla \frac{1}{2} J(\boldsymbol{w}) = \mathbf{0} = \sum_{i=1}^{\ell} y_i \boldsymbol{x}_i - \left[\sum_{i=1}^{\ell} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}} \right] \boldsymbol{w} \ . \tag{3.9}$$

To see this, note that each term in the sum $\sum_{i=1}^{\ell} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}}$ is an outer product of the form

$$\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} = \begin{pmatrix} x_1 x_1 & x_1 x_2 & \ldots & x_1 x_n \\ x_2 x_1 & x_2 x_2 & \ldots & x_2 x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n x_1 & x_n x_2 & \ldots & x_n x_n \end{pmatrix} \ . \tag{3.10}$$

Furthermore, it holds $\sum_{i=1}^{\ell} y_i \boldsymbol{x}_i = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y}$ and $\sum_{i=1}^{\ell} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}} = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}$. Thus, we have to solve:

$$\mathbf{0} = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{w} \tag{3.11}$$

This equation implies $\boldsymbol{X}\boldsymbol{y} = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{w}$ and we get the solution

$$\boldsymbol{w}^{\star} = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \ . \tag{3.12}$$

The matrix $\boldsymbol{X}^{\dagger} = \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \right)^{-1} \boldsymbol{X}^{\mathrm{T}}$ is called Moore-Penrose pseudo-inverse.

3.2.2 Sum-of-squares error and Gaussian Noise*

Why do we minimize the sum-of-squares error? Intuitively, because it has nice properties. First of all, it is differentiable. Second, it penalizes large deviations from a target value more than proportionally stronger than small deviations.

However, the sum-of-squares error can also be derived from a maximum-likelihood estimation of the parameters under the assumption of normally distributed noise (see Appendix A.1.2).

Let us consider a deterministic target functions $f : \mathcal{X} \to \mathbb{R}$ perturbed by zero-mean additive Gaussian noise with variance σ^2, then we have

$$p(y \,|\, x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - f(x))^2}{2\sigma^2}} \ . \tag{3.13}$$

That is, $p(y \,|\, x)$ is normally distributed with mean $f(x)$ and variance σ^2.

Given $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ the likelihood of the model $h(x) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$ is

$$p((y_1,\ldots,y_\ell)^{\mathrm{T}} \mid (x_1,\ldots,x_\ell),\boldsymbol{w},\sigma^2) = \prod_{i=1}^{\ell} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_1-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)^2}{2\sigma^2}} \quad . \tag{3.14}$$

For convenience, we consider the logarithm turning products into sums and get

$$\begin{aligned}
\ln p(\boldsymbol{y} \mid \boldsymbol{w}) &= \ln \prod_{i=1}^{\ell} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)^2}{2\sigma^2}} \\
&= \sum_{i=1}^{\ell} \ln \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)^2}{2\sigma^2}} \\
&= -\frac{\ell}{2}\ln\sigma^2 - \frac{\ell}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{\ell}\left\{y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}^2 \quad .
\end{aligned} \tag{3.15}$$

The first two terms do not depend on $\boldsymbol{w}$ and thus just the final sum has to be considered when maximizing w.r.t. the model parameters. Thus, maximizing the likelihood corresponds to minimizing the sum-of-squares loss under the assumption of Gaussian noise.

3.3 Overfitting

Let us consider the case $\ell \leq n$. That is, we have no more data points than features and therefore more model parameters $(n+1)$ than sample data points. In this case, linear regression will always find a parameter vector $\boldsymbol{w}$ with zero error $J = 0$ on the training data by simply solving the system of linear equations

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} \quad . \tag{3.16}$$

This is possible even if the targets y_i are independent of the corresponding $\boldsymbol{x}_i$, for example, if all x_{ij} and y_i are drawn independently at random. That is, we can derive a model describing a relation beween the $\boldsymbol{x}_i$ and the corresponding y_i with a perfect fit to the data although there is absolutely no such relationship. This happens because our model is too flexible (or too complex) – there is not enough data to reliably estimate the parameters without further constraints. This effect is called overfitting.

Insight 3.1. Overfitting occurs if the model faithfully reflects idiosyncrasies of the training data rather than the underlying process that has generated the data. This may happen if the model is too complex in relation to the amount of available information.

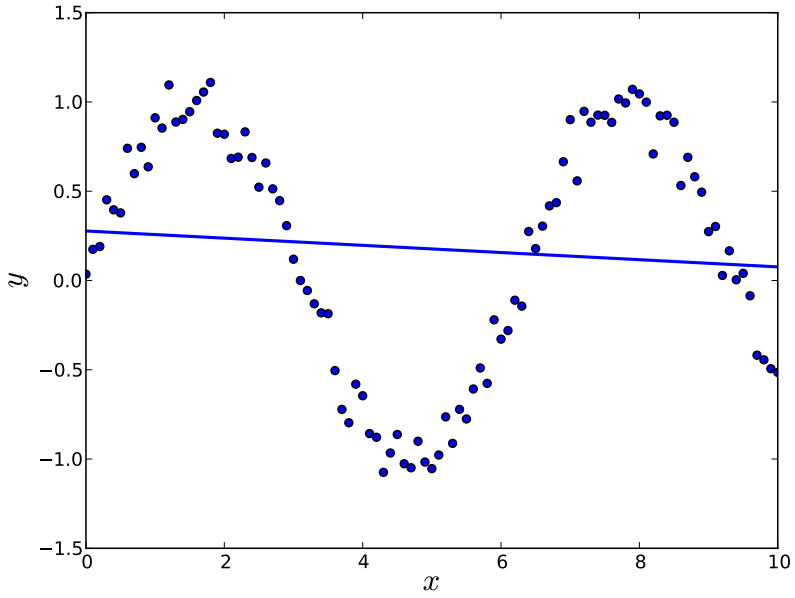

Fig. 3.2. Linear regression trying to fit a sine function that has been perturbed by noise.

Overfitting can be viewed as the opposite of the effect shown in Figure 3.2. This figure showns an example where the model is not flexible enough to fit the given data.

Our definition of overfitting and complexity or flexibility of models is very hand-waving. More formal definitions exist, but explaining them is unfortunately beyond this minilecture.

4

k-means Clustering

The goal of clustering or segmentation is to assign data points (e.g., records in a database) to groups or clusters. Similar points should be in the same cluster, dissimilar points should be in different clusters. In *hard clustering* each data point belongs exactly to one group, while in *soft clustering* a data point can belong to more than one group. The arguably most fundamental segmentation algorithm is k-means clustering [e.g., Lloyd, 1982].

4.1 Description of the algorithm

Consider we are given the observations $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$, $\boldsymbol{x}_i \in \mathbb{R}^n$, $i \in 1, \ldots, \ell$. Note that there is no notion of a target, a class label, or dependent variables. Therefore, clustering is referred to as an *unsupervised learning* method in the machine learning literature.

The *k-means clustering* algorithm (k-means for short) is a hard clustering method dividing the data into k clusters, where the number k has to be chosen *a priori*. Each cluster i is identified by a single point $\boldsymbol{\mu}_i \in \mathbb{R}^n$, the *cluster mean* or *cluster centroid*. Similarity is measured by the Euclidean distance, see Example 2.1 in Chapter 2.

Let all data points assigned to a cluster k be in the set S_k. Because we assign each data point to exactly one cluster we have $S = \bigcup_{i=1}^{k} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. The goal of k-means – the underlying objective function – is to minimize the cost function or *distortion measure*

$$J = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \tag{4.1}$$

by finding appropriate partitions $S_1, \ldots, S_k$ and cluster centroids $\boldsymbol{\mu}_i$ ($i = 1, \ldots, k$). To optimize (4.1), the k-means algorithm iterates the following two steps:

Data assignment*:* Assign each data point to the cluster represented by the centroid which is most similar to the data point. This leads to a new partitioning of the data.

Centroid relocation*:* Recompute the cluster centroids by setting them equal to the mean of all data points assigned to the respective cluster.

The algorithm stops when the partitioning has not changed in two subsequent data assignment steps. It is important to note that the algorithm may converge in a local optimum that is not the global minimum of J. Clustering using k-means is outlined in Algorithm 4.

Algorithm 4: k-means clustering

Input: $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$, number of clusters k
Output: cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, partitioning of the data $S_1, \ldots, S_k$
initialize class centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
repeat
  $\forall i = 1, \ldots, k : S'_i \leftarrow S_i$
  `/* data assignment, ties are broken at random or by deterministic rule */`
  $\forall i = 1, \ldots, k : S_i \leftarrow \{\boldsymbol{x} \,|\, \boldsymbol{x} \in S \wedge i = \operatorname{argmin}_j \|\boldsymbol{\mu}_j - \boldsymbol{x}\|\}$
  `/* centroid relocation */`
  $\forall i = 1, \ldots, k : \boldsymbol{\mu}_i \leftarrow \frac{1}{|S_i|} \sum_{\boldsymbol{x} \in S_i} \boldsymbol{x}$
until $\forall i = 1, \ldots, k : S'_i = S_i$
Result: cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, partitioning of the data $S_1, \ldots, S_k$

Note that Algorithm 4 starts with initializing the cluster centroids. This initialization is extremely important. It determines how long it takes until the algorithms converges as well as the final solution we get in the end. The most common way to initialize the k vectors $\boldsymbol{\mu}_i$ is by using randomly chosen points from S. This is usually preferable to choosing k random points from $\mathbb{R}^n$. If possible, the initial class means should be set by an educated guess, which may, for example, stem from previous segmentations or clustering with a subset of the data (which can be done efficiently and may speed-up the clustering of the full data set).

There is no single "best" clustering algorithm, and there is even no "best" k-means clustering solution. In practice, often several clustering results – produce by the same method or different algorithms – are produced. These are inspected by an expert, who chooses the final clustering Ghosh and Liu [2009].

4.2 Deriving the algorithm

In this section, we briefly show that the iterative Algorithm 4 indeed minimizes the distortion measure (4.1). The first of the two iterated steps, assigning the

points to the clusters, minimizes J w.r.t. the partitioning keeping the cluster centroids fixed. For fixed cluster centroids, it is clear that a vector $\boldsymbol{x}$ should be assigned to the nearest cluster i with mean $\boldsymbol{\mu}_i$, because $\|\boldsymbol{\mu}_j - \boldsymbol{x}\|$ can not be smaller than $\|\boldsymbol{\mu}_i - \boldsymbol{x}\|$ and thus assigning to cluster j could only increase J.

The second step, centroid relocation, minimizes J w.r.t. to the cluster means keeping the partitioning fixed. Let μ_{ij} and x_j be the jth component of $\boldsymbol{\mu}_i$ and $\boldsymbol{x}$, respectively. The partial derivative of (4.1) w.r.t. μ_{ij} is

$$\frac{\partial J}{\partial \mu_{ij}} = -2 \sum_{\boldsymbol{x} \in S_i} (x_j - \mu_{ij}) \tag{4.2}$$

Setting the derivative to zero gives

$$\boldsymbol{\mu}_i = \frac{1}{|S_i|} \sum_{\boldsymbol{x} \in S_i} \boldsymbol{x} \ . \tag{4.3}$$

Each partitioning uniquely defines the cluster means. For deterministic breaking of ties, each set of cluster means implies a particular partitioning. Thus, once the partitioning does not change after a relocation of the centroids the algorithm has converged.

5

Principal Component Analysis

Principal component analysis (PCA), also known as Karhunen-Loève transform, is arguably the most fundamental technique in *unsupervised learning*. It is frequently used for (linear) dimensionality reduction, (lossy) data compression, feature extraction, and data visualization.

Let us consider a set of ℓ data points $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$ with $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell \in \mathcal{X} = \mathbb{R}^n$. We use n real values (n features) to represent a single data point. How can we reduce the length of the description to $m < n$ variables such that as much information as possible is preserved? How many dimensions are needed to capture a certain percentage of the variability of the data? These are typical questions that arise when we want to do dimensionality reduction, feature selection, and compression (these three terms usually refer to similar processes, emphasizing different aspects of the same algorithmic procedure). Dimensionality reduction is closely linked to visualization. When visualizing high-dimensional data, the question arises of how to project the data to two or three dimensions such that the visualization of these dimensions reflects as much of the variability of the data as possible.

Principle component analysis gives – under particular assumptions – answers to these questions. Because PCA is so relevant, it is important to understand it in detail, which requires some linear algebra (in fact, deriving PCA is a very nice linear algebra revision course). In the following, I will first outline the PCA algorithm and then provide the mathematical background. Large parts follow the derivation by Bishop [2006].[1]

5.1 Description of the algorithm

The goal of PCA is to find an m-dimensional affine linear model of the n-dimensional data S that represents the original data as accurately as possible.

[1] Bishop [2006] denotes the number of data points by N, the dimension of the input space by D and the reduced dimension by M. We use ℓ, n, and m instead.

Such a model mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$ has the general form

$$f_{\text{dec}}(\boldsymbol{z}) = \boldsymbol{b} + \boldsymbol{U}_m \boldsymbol{z} \ , \tag{5.1}$$

where $\boldsymbol{z} \in \mathbb{R}^m$, $\boldsymbol{b} \in \mathbb{R}^n$, and $\boldsymbol{U}_m \in \mathbb{R}^{n \times m}$. We identify the columns of $\boldsymbol{U}_m$ by the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in \mathbb{R}^n$ and require that these vectors are pairwise orthogonal and have unit length. This means $\boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{u}_j = 1$ if $i = j$ and $\boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{u}_j = 0$ otherwise. The model represents data in the n-dimensional $\mathbb{R}^n$ by m-dimensional parameters $\boldsymbol{z}$. Equation (5.1) defines an m-dimensional *affine subspace* also called an m-dimensional *linear manifold*. In the $\mathbb{R}^n$, one- and two-dimensional linear manifolds are standard lines and planes.

The model is referred to as f_{dec}, because it can be interpreted as decoding a m-dimensional representation (encoding) of a n-dimensional data record. The corresponding affine linear function encoding a n-dimensional data record will be denoted by $f_{\text{enc}} : \mathbb{R}^n \to \mathbb{R}^m$, accordingly.

We measure the quality of the model by the *reconstruction error* defined as the sum-of-squares error

$$J = \frac{1}{\ell} \sum_{i=1}^{\ell} \|\boldsymbol{x}_i - f_{\text{dec}}(\boldsymbol{z}_i)\|^2 \tag{5.2}$$

between the data points $\boldsymbol{x}_i$ and their approximation by the model $\boldsymbol{z}_i$. Finding the optimal model corresponds to solving

$$\min_{\boldsymbol{b}, \boldsymbol{U}_m, \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell\}} \frac{1}{\ell} \sum_{i=1}^{\ell} \|\boldsymbol{x}_i - f_{\text{dec}}(\boldsymbol{z}_i)\|^2 \tag{5.3}$$

subject to the required constraints on $\boldsymbol{U}_m$.

It turns out that the optimal choice for $\boldsymbol{b}$ is the empirical mean

$$\overline{\boldsymbol{x}} = \frac{1}{\ell} \sum_{i=1}^{\ell} \boldsymbol{x}_i \tag{5.4}$$

and that the data points should be encoded by

$$\boldsymbol{z}_i = \boldsymbol{U}_m^{\mathrm{T}} (\boldsymbol{x}_i - \overline{\boldsymbol{x}}) \ , \tag{5.5}$$

that is

$$f_{\text{enc}}(\boldsymbol{x}) = \boldsymbol{U}_m^{\mathrm{T}} (\boldsymbol{x} - \overline{\boldsymbol{x}}) \ . \tag{5.6}$$

Finally, the m columns of $\boldsymbol{U}_m$ should correspond to the first m eigenvectors of the *data covariance matrix* or *empirical covariance matrix*

$$\boldsymbol{S} = \frac{1}{\ell} \sum_{i=1}^{\ell} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathrm{T}} \ , \tag{5.7}$$

where we assume that the eigenvectors are ordered by decreasing eigenvalue. These vectors are called the first m *principal components*. The ordering by eigenvalue ensures that first m basis vectors correspond to the best m-dimensional manifold in the context of the questions posed above. Note that (5.7) is a sum of *outer products* (e.g., see equation 3.10) and therefore a sum of matrices. Because it estimates the covariance of the distribution generating S, $\boldsymbol{S}$ is also called the *empirical covariance matrix*.

It further turns out that the model does not only minimize the reconstruction error, it is also the affine linear model that yields the representation that maximizes the overall variance of the points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell$ (see Appendix A.1.3 for notes about variance and covariance). That is, the model preserves as much variability as possible when representing S. We measure the variability of S by the trace $\operatorname{tr}\{\boldsymbol{S}\} = \sum_{i=1}^{n} s_{ii}$ and the variability of the $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell$ accordingly (see sections 5.2.3 and 5.2.4) by

$$\frac{1}{\ell} \sum_{i=1}^{\ell} \left[\boldsymbol{U}_m^{\mathrm{T}} \boldsymbol{x}_i - \boldsymbol{U}_m^{\mathrm{T}} \overline{\boldsymbol{x}}\right] \left[\boldsymbol{U}_m^{\mathrm{T}} \boldsymbol{x}_i - \boldsymbol{U}_m^{\mathrm{T}} \overline{\boldsymbol{x}}\right]^{\mathrm{T}} = \sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j \quad . \tag{5.8}$$

Algorithm 5: dimensionality reduction using PCA

Input: data $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$, number of dimensions m

1 compute the empirical mean $\overline{\boldsymbol{x}} = \frac{1}{\ell} \sum_{i=1}^{\ell} \boldsymbol{x}_i$

2 compute empirical covariance matrix $\boldsymbol{S} = \frac{1}{\ell} \sum_{i=1}^{\ell} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathrm{T}}$

3 compute the $n \times m$ matrix $\boldsymbol{U}_m$ composed of the first m eigenvectors of $\boldsymbol{S}$, where the eigenvectors are ordered by decreasing eigenvalue

4 compute $\boldsymbol{z}_i = \boldsymbol{U}_m^{\mathrm{T}} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})$ for $i = 1, \ldots, \ell$

Output: mean $\overline{\boldsymbol{x}}$, principal components $\boldsymbol{U}_m$, projected data $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell\}$, decoding model $f_{\mathrm{dec}}(\boldsymbol{z}) = \overline{\boldsymbol{x}} + \boldsymbol{U}_m \boldsymbol{z}$ and encoding model $f_{\mathrm{enc}}(\boldsymbol{x}) = \boldsymbol{U}_m^{\mathrm{T}} (\boldsymbol{x} - \overline{\boldsymbol{x}})$

Algorithm 5 summarizes PCA. Given data, it computes the optimal linear manifold in terms of reconstruction error and variance maximization for a given dimensionality m. It also returns the low dimensional representations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell$ of the data points S. Because the $\boldsymbol{z}_i$ result from projecting the zero mean data onto the principal components they are referred to as the projected data.

Example 5.1 (Two-dimensional Gaussian distribution). Figure 5.1 shows an example data set drawn from a multi-variate Gaussian distribution. The original data S are the blue dots. The red arrows indicate the two principal components (scaled with the square root of the corresponding eigenvalues). If we fit a one-dimensional ($m = 1$) PCA model to the data, we get the green dots (the $\boldsymbol{z}_i$). In this example, PCA nicely reflects the structure of the data.

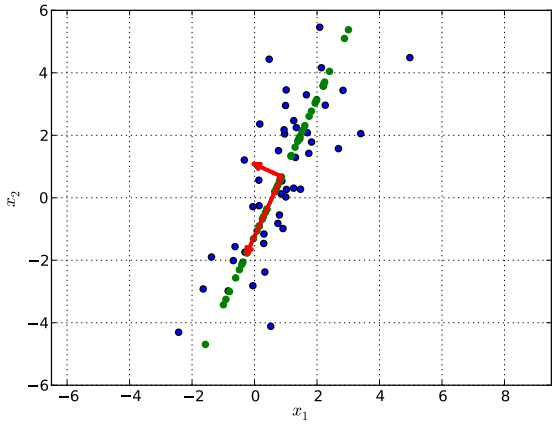

Fig. 5.1. Two-dimensional PCA example. The blue dots are the raw input data S. The red arrows indicate the two principal components (scaled with the square root of the corresponding eigenvalues). If the input data is encoded by the first principal component and is then reconstructed, we get the green dots (i.e., the $\boldsymbol{z}_i$).

This first principal component clearly corresponds to the main direction along which the variance is maximal.

Example 5.2 (Face database). Let us consider the *Cambridge Face Database* [Samaria and Harter, 1994] as an example. It contains 92×122 images of frontal faces, some examples are shown in Figure 5.2. We can represent each face by a vector by inflating the image, that is, by concatenating the image rows. Although the images are rather small, we get a 10304-dimensional representation (i.e., $n = 10304$ and $\boldsymbol{x}_i \in \mathbb{R}^{10304}$). That is, in the original representation 10304 basis vectors, one for each pixel, define the image space. Principal component analysis can help us to significantly reduce the description length of the images.

The mean of the Cambridge Face Database is depicted in Figure 5.3.

The first ten eigenvectors of the Cambridge Face Database are shown in Figure 5.4. If we depict the eigenvectors as images (by reversing the transformation that turns the images into vectors), we get the so called "eigenfaces". Using an eigenface representation is a popular technique for face recognition [Turk and Pentland, 1991].

Figure 5.5 illustrates what is meant by a representation that tries to preserve as much variance in the data as possible. If we use the first two eigenvectors to represent the faces and plot the resulting projected data, we get the left plot in Figure 5.5. This representation maximizes the variance, as for

Fig. 5.2. Examples from the Cambridge Face Database [Samaria and Harter, 1994].

Fig. 5.3. The mean face ($\overline{\boldsymbol{x}}$).

example compared to the right plot showing the projection onto the 99th and 100th eigenvector.

The more principal components we use in our model, the smaller the reconstruction error. This is shown in Figure 5.6, where a face is reconstructed from a 300, 200, and 100 dimensional PCA model.

Principal component analysis relies on an affine linear model of the data. If the structure of the data is highly non-linear, PCA will not reveal it and the PCA model will be not appropriate.

Fig. 5.4. First eight "eigenfaces", ordered from top left to bottom right.

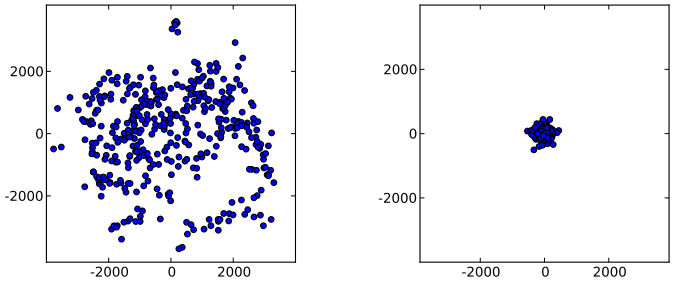

Fig. 5.5. The 400 faces in the Cambridge Face Database encoded by the first two principal components (left) and by the 99th and 100th principal component (right).

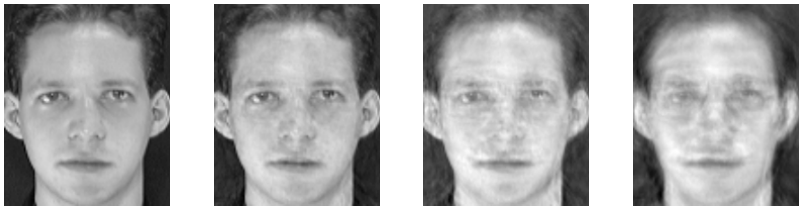

Fig. 5.6. A sample face reconstructed from all, the first 300, the first 200, and the first 100 principal components (from left to right).

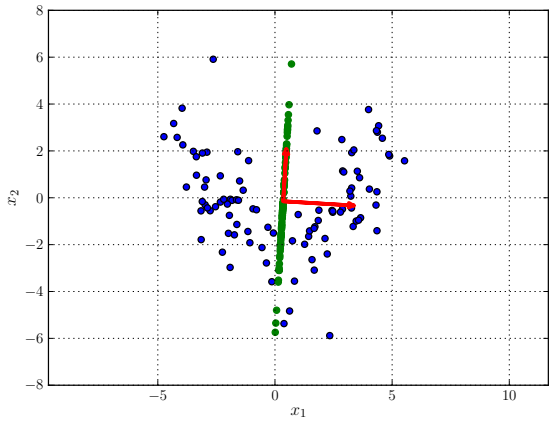

Fig. 5.7. A data set not ideally suited for PCA.

Example 5.3. As a negative example, let us consider Figure 5.7. It shows data drawn from two multi-variate Gaussian distributions. Again, the original data S are the blue dots, the red arrows indicate the two principal components, and the green dots the projected data. Here the principal components may be regarded as not a proper description of the data set.

5.2 Deriving PCA

This section justifies the PCA procedure outlined in the previous section. It starts with some general comments on orthogonal bases and coordinate systems. Then we derive the PCA procedure in 3 steps:

1. It is shown that using (5.4) and (5.5) in the model indeed minimizes the reconstruction error (5.2) for a given $\boldsymbol{U}_m$.
2. The equivalence between variance maximization and reconstruction error minimization is established.
3. It is proven that the variance is maximized by considering the first principal components.

5.2.1 Bases and coordinate systems

Principal component analysis is best understood as a change of the basis (i.e., a change of the coordinate system). Let the new basis be given by the n basis vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$. Principal component analysis constructs an *orthonormal* basis, which means that the vectors $\boldsymbol{u}_i$ all have unit length, that

is, $\|\boldsymbol{u}_1\|_2 = 1$ or equivalently $\boldsymbol{u}_1^{\mathrm{T}}\boldsymbol{u}_1 = 1$, and that they are pairwise *orthogonal*. The latter means we have $\boldsymbol{u}_i^{\mathrm{T}}\boldsymbol{u}_j = 0$ for $i \neq j$. What does changing the coordinate system mean? Think of a vector $\boldsymbol{x} \in \mathbb{R}^2$. It can be expressed in the standard coordinate system with basis vectors $(0,1)^{\mathrm{T}}$ and $(1,0)^{\mathrm{T}}$ by $\boldsymbol{x} = (1,0)^{\mathrm{T}}x_1 + (0,1)^{\mathrm{T}}x_2$. We may read this as a description of the "way from $\mathbf{0}$ to $\boldsymbol{x}$: First go x_1 units in the direction $(1,0)$ and then x_2 units in direction $(1,0)$." Changing the basis now means "expressing the way using two other, non-parallel directions $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$: First go z_1 units in the direction $\boldsymbol{u}_1$ and then z_2 units in direction $\boldsymbol{u}_2$" (i.e., $\boldsymbol{x} = \sum_{i=1}^{2} z_i\boldsymbol{u}_i$).

We gather basis vectors $\mathbb{R}^n$ in an $n \times n$ matrix $\boldsymbol{U}$ such that the columns of $\boldsymbol{U}$ correspond to the basis vectors. The ith column of $\boldsymbol{U}$ is given by $\boldsymbol{u}_i$. A square matrix composed of an orthonormal basis is an *orthogonal* matrix having the property $\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{U}^{-1}$. We now define the matrix $\boldsymbol{U}_m$ as the $n \times m$ matrix gathering just the first m basis vectors. Accordingly, for $i \leq m$ the ith column of $\boldsymbol{U}_m$ is given by $\boldsymbol{u}_i$ and the ith row of $\boldsymbol{U}^{\mathrm{T}}$ by $\boldsymbol{u}_i^{\mathrm{T}}$.

That is, $\boldsymbol{z}_i = (z_{i1}, \ldots, z_{im})^{\mathrm{T}}$ results from projecting the vector $\boldsymbol{x}_i - \overline{\boldsymbol{x}}$ onto the m vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$. For the jth component z_{ij} of $\boldsymbol{z}_i$ we have $z_{ij} = \boldsymbol{u}_j^{\mathrm{T}}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})$.

5.2.2 Optimal reconstruction

First, we have to realize that every point $\boldsymbol{x}_i$ can be expressed in terms of a complete orthonormal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_\ell\}$ by

$$\boldsymbol{x}_i = \sum_{j=1}^{n} \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{u}_j\right)\boldsymbol{u}_j \ . \tag{5.9}$$

The $\left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{u}_j\right)\boldsymbol{u}_j$ are the projections of $\boldsymbol{x}_i$ onto the basis vectors $\boldsymbol{u}_j$ and the new representation results from summing up these projections. The correctness of (5.9) can be proved by making the ansatz

$$\boldsymbol{x}_i = \sum_{j=1}^{n} \alpha_{ij}\boldsymbol{u}_j \ . \tag{5.10}$$

Because the $\boldsymbol{u}_j$ form a complete basis, for each $\boldsymbol{x}_i$ parameters α_{ij} must exist such that this equality holds true. If we multiply (5.10) from the left with some transposed basis vector $\boldsymbol{u}_j^{\mathrm{T}}$, we get $\boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{x}_i = \sum_{l=1}^{n} \alpha_{il}\boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{u}_l = \alpha_{ij}\boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{u}_j = \alpha_{ij}$ because the basis vectors are pairwise orthogonal and have unit length.

Given our basis, what is the best way – in the sense of minimizing the reconstruction error J – to represent a vector $\boldsymbol{x}_i$ by just the first m basis vectors? Such a representation has the general form

$$f_{\mathrm{dec}}(\boldsymbol{z}_i) = \underbrace{\sum_{j=1}^{m} z_{ij}\boldsymbol{u}_j}_{\text{dependent on } i} + \underbrace{\sum_{j=m+1}^{n} b_j\boldsymbol{u}_j}_{\text{independent of } i} \ . \tag{5.11}$$

Coming back the metaphor used before, this means describing a way by a limited set of directions (first sum) from a fixed starting point (the second sum). For a fixed basis, we are free to choose the z_{ij} and b_j in the representation (5.11) to reduce the distortion measure J. Plugging (5.11) into (5.2) we get

$$J = \frac{1}{\ell} \sum_{i=1}^{\ell} \left\| \boldsymbol{x}_i - \sum_{j=1}^{m} z_{ij} \boldsymbol{u}_j - \sum_{j=m+1}^{n} b_j \boldsymbol{u}_j \right\|^2 \quad . \tag{5.12}$$

To find the optimal z_{ij}, we check when derivative $\partial J / \partial z_{ij}$ vanishes, which happens for $z_{ij} = \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{x}_i$:

$$\begin{aligned} \frac{\partial J}{\partial z_{ij}} &= \frac{2}{\ell} \sum_{i=1}^{\ell} \boldsymbol{u}_j^{\mathrm{T}} \left(\boldsymbol{x}_i - \sum_{k=1}^{m} z_{ik} \boldsymbol{u}_k - \sum_{k=m+1}^{n} b_k \boldsymbol{u}_j \right) \\ &= \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{x}_i - z_{ij} \quad . \end{aligned} \tag{5.13}$$

Here we used the fact that $\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{u}_k$ is one for $k = j$ and zero otherwise and basic rules of derivation (such as $\frac{\partial}{\partial \boldsymbol{a}} \boldsymbol{a}^{\mathrm{T}} \boldsymbol{b} = \boldsymbol{b}^{\mathrm{T}}$). In the same way we derive $b_j = \overline{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{u}_j$.

Substituting the z_{ij} and b_j in (5.11) we arrive at

$$f_{\mathrm{dec}}(\boldsymbol{z}_i) = \sum_{j=1}^{m} (\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{x}_i) \boldsymbol{u}_j + \sum_{j=m+1}^{n} (\overline{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{u}_j) \boldsymbol{u}_j \tag{5.14}$$

$$= \overline{\boldsymbol{x}} + \sum_{j=1}^{m} (\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{u}_j - \overline{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{u}_j) \boldsymbol{u}_j = \overline{\boldsymbol{x}} + \boldsymbol{U}_m \underbrace{\boldsymbol{z}_i}_{\boldsymbol{U}_m^{\mathrm{T}} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})} \quad . \tag{5.15}$$

5.2.3 Equivalence of reducing the reconstruction error and maximizing the variance*

Now we show that minimizing the distortion measure J is equal to choosing $\boldsymbol{U}_m$ maximizing the measure of variance

$$\sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j \quad . \tag{5.16}$$

Using equations (5.9) and (5.14) we get

$$\begin{aligned} \boldsymbol{x}_i - f_{\mathrm{dec}}(\boldsymbol{z}_i) &= \sum_{j=1}^{n} \left(\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{u}_j \right) \boldsymbol{u}_j - \sum_{j=1}^{m} (\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{x}_i) \boldsymbol{u}_j - \sum_{j=m+1}^{n} (\overline{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{u}_j) \boldsymbol{u}_j \\ &= \sum_{j=m+1}^{n} \left[(\boldsymbol{x}_j - \overline{\boldsymbol{x}})^{\mathrm{T}} \boldsymbol{u}_j \right] \boldsymbol{u}_j = \sum_{j=m+1}^{n} \left(\boldsymbol{x}_j^{\mathrm{T}} \boldsymbol{u}_j - \overline{\boldsymbol{x}}^{\mathrm{T}} \boldsymbol{u}_j \right) \boldsymbol{u}_j \quad . \end{aligned} \tag{5.17}$$

Thus

$$\|\boldsymbol{x}_i - f_{\text{dec}}(\boldsymbol{z}_i)\|^2 = \left[\sum_{j=m+1}^{n} \left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{u}_j - \overline{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{u}_j\right)\boldsymbol{u}_j\right]^{\mathrm{T}} \left[\sum_{j=m+1}^{n} \left(\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{u}_j - \overline{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{u}_j\right)\boldsymbol{u}_j\right]$$
$$= \sum_{j=m+1}^{n} (\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{u}_j - \overline{\boldsymbol{x}}\boldsymbol{u}_j)^2 \ . \quad (5.18)$$

The last equality holds because the terms involving $\boldsymbol{u}_i^{\mathrm{T}}$ and $\boldsymbol{u}_j$ vanish for $j \neq i$ and $\boldsymbol{u}_i^T\boldsymbol{u}_i = 1$. Using the derived equality the error can be written as

$$J = \frac{1}{\ell}\sum_{i=1}^{\ell}\sum_{j=m+1}^{n} (\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{u}_j - \overline{\boldsymbol{x}}\boldsymbol{u}_j)^2 = \sum_{j=m+1}^{n} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j \ . \quad (5.19)$$

Therefore minimizing J is equal to maximizing $\sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j$, because we have from the cyclic property of the matrix trace

$$\sum_{j=1}^{n} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j = \mathrm{tr}\{\boldsymbol{U}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{U}\} = \mathrm{tr}\{\boldsymbol{S}\} \quad (5.20)$$

independent of $\boldsymbol{U}$ and therefore

$$\sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j = \text{const} - \sum_{j=m+1}^{n} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j \ . \quad (5.21)$$

5.2.4 Maximizing the Variance*

From the empirical covariance matrix

$$\boldsymbol{S} = \frac{1}{\ell}\sum_{i=1}^{\ell}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathrm{T}} \quad (5.22)$$

we derive $\sum_{j=1}^{n} \boldsymbol{u}_j^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_j = \mathrm{tr}\{\boldsymbol{S}\}$ as a measure of variability of the data. Accordingly, we have the empirical $m \times m$ covariance matrix of our projected data

$$\frac{1}{\ell}\sum_{i=1}^{\ell} \left[\boldsymbol{U}_m^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{U}_m^{\mathrm{T}}\overline{\boldsymbol{x}}\right]\left[\boldsymbol{U}_m^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{U}_m^{\mathrm{T}}\overline{\boldsymbol{x}}\right]^{\mathrm{T}} =$$
$$\frac{1}{\ell}\sum_{i=1}^{\ell} \boldsymbol{U}_m^{\mathrm{T}}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})\left[\boldsymbol{U}_m^{\mathrm{T}}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})\right]^{\mathrm{T}} =$$
$$\frac{1}{\ell}\sum_{i=1}^{\ell} \boldsymbol{U}_m^{\mathrm{T}}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathrm{T}}\boldsymbol{U}_m = \boldsymbol{U}_m^{\mathrm{T}}\boldsymbol{S}\boldsymbol{U}_m \quad (5.23)$$

and ask for $\boldsymbol{U}_m$ maximizing the variability

$$\sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j \tag{5.24}$$

of the projected data.

Let us first consider the case $m = 1$. That is, we want to project n-dimensional data to a 1-dimensional subspace, that is, a line in $\mathbb{R}^n$. The direction of this line is denoted by $\boldsymbol{u}_1$. For describing a direction, the length of $\boldsymbol{u}_1$ does not matter, and we require without loss of generality that $\boldsymbol{u}_1$ is a unit vector. So, we ask for the direction $\boldsymbol{u}_1$ maximizing $\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_1$ under the constraint that $\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{u}_1 = 1$. Using our considerations from Appendix A.3 and solving $\max_{\boldsymbol{u}_1} \boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_1$ subject to the constraint $c(\boldsymbol{u}_1) = \boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{u}_1 - 1 = 0$, we get the condition

$$\boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1 \ . \tag{5.25}$$

That is, $\boldsymbol{u}_1$ must be an eigenvector of S with eigenvalue λ_1. When we multiply the above equation with $\boldsymbol{u}_1^{\mathrm{T}}$ from the left and use $\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{u}_1 = 1$, we see that the variance $\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_1$ is given by

$$\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \ . \tag{5.26}$$

The general case $m > 1$ is proven by induction [Bishop, 2006]. The previous result for $m = 1$ is the base case. For the inductive step we assume that the first k eigenvectors of S are the columns of $\boldsymbol{U}_k$ maximizing the variance of the projected data using a k-dimensional PCA model. We want to solve $\max_{\boldsymbol{u}_{k+1}} \sum_{j=1}^{k+1} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j$, which is equivalent to solving $\max_{\boldsymbol{u}_{k+1}} \boldsymbol{u}_{k+1}^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_{k+1}$. We have several equality constraints. As above, we have $c_{k+1}(\boldsymbol{u}_{k+1}) = \boldsymbol{u}_{k+1}^{\mathrm{T}} \boldsymbol{u}_{k+1} - 1 = 0$. We seek an orthogonal basis, therefore we additionally demand $c_k(\boldsymbol{u}_{k+1}) = \boldsymbol{u}_{k+1}^{\mathrm{T}} \boldsymbol{u}_k = 0$ for $1 \leq k \leq m$. This ensures that the new vector is orthogonal to and thus independent of the others. Setting the derivative of the objective function w.r.t. $\boldsymbol{u}_k$ equal to the sum of the derivatives of the equality constraints times corresponding Lagrange multipliers $(\lambda_{k+1}, \eta_1, \ldots, \eta_k)$ yields

$$2\boldsymbol{S} \boldsymbol{u}_{k+1} = 2\boldsymbol{u}_{k+1} \lambda_{k+1} + \sum_{i=1}^{k} \eta_i \boldsymbol{u}_i \ . \tag{5.27}$$

Multiplying with any $\boldsymbol{u}_j^{\mathrm{T}}$, $j = 1, \ldots, k$, from the left

$$2 \underbrace{\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_{k+1}}_{=\lambda_j \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{u}_{k+1} = 0} = 2\lambda_{k+1} \underbrace{\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{u}_{k+1}}_{=0} + \underbrace{\sum_{i=1}^{k} \eta_i \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{u}_i}_{=\eta_j} \tag{5.28}$$

implies that all η_j are zero. Note that $\boldsymbol{u}_j$ and $\boldsymbol{u}_{k+1}$ are orthogonal and that $\boldsymbol{u}_j$ is an eigenvector of $\boldsymbol{S}$ implying $\boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} = \lambda_j \boldsymbol{u}_j^{\mathrm{T}}$. Setting all η_j in (5.28) to zero

yields $\boldsymbol{S}\boldsymbol{u}_{k+1} = \lambda_{k+1}\boldsymbol{u}_{k+1}$. That is, $\boldsymbol{u}_{k+1}$ is an eigenvector of $\boldsymbol{S}$. Multiplying (5.27) with $\boldsymbol{u}_{k+1}^{\mathrm{T}}$ from the left

$$2\boldsymbol{u}_{k+1}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_{k+1} = 2\lambda_{k+1}\underbrace{\boldsymbol{u}_{k+1}^{\mathrm{T}}\boldsymbol{u}_{k+1}}_{=1} + \sum_{i=1}^{k}\eta_i\underbrace{\boldsymbol{u}_{k+1}^{\mathrm{T}}\boldsymbol{u}_i}_{=0} \tag{5.29}$$

implies that the "added" variance is given by

$$\boldsymbol{u}_{k+1}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{u}_{k+1} = \lambda_{k+1} \ . \tag{5.30}$$

Therefore the overall variance is maximized by adding the eigenvector with the largest eigenvalue among those not already selected. This completes the inductive step.

5.3 PCA for high dimensional data and small samples

The time complexity of computing the eigenvalue decomposition of an n times n matrix is cubic in n. The number n of dimensions easily gets very large, for example, if we want to analyze images. Let us consider the – comparatively small – 92×122 images in the Cambridge face database (see Example 5.2). We can apply PCA in a straight forward way to images if we inflate the image and represent it as a vector by concatenating the rows. Each face in the dataset is then represented by a 10304 dimensional vector and the corresponding data covariance matrix has 106172416 entries.

Let us write the data covariance matrix as

$$\boldsymbol{S} = \boldsymbol{X}_0^{\mathrm{T}}\boldsymbol{X}_0 \ , \tag{5.31}$$

where the $\ell \times n$ data matrix $\boldsymbol{X}_0$ is composed of the training data points after subtracting the mean such that the ith row corresponds to $(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathrm{T}}$.

Now let ℓ be smaller than n. In the following, it is shown how to compute the ℓ principal components of the $n \times n$ matrix $\boldsymbol{S}$ by working with an $\ell \times \ell$ matrix. For $\ell \ll n$, this leads to a drastic speed-up.

The restriction to the first ℓ principal components is no limitation. Basic linear algebra teaches us that the rank of $\boldsymbol{S}$ and the number of non-zero eigenvalues is not larger than ℓ anyway.

Let us consider the $\ell \times \ell$ matrix $\boldsymbol{X}_0\boldsymbol{X}_0^{\mathrm{T}}$. Given an eigenvector $\boldsymbol{v}$ and eigenvalue λ of this smaller matrix, multiplying

$$\boldsymbol{X}_0\boldsymbol{X}_0^{\mathrm{T}}\boldsymbol{v} = \lambda\boldsymbol{v} \tag{5.32}$$

with $\boldsymbol{X}_0^{\mathrm{T}}$ from the left yields

$$\boldsymbol{X}_0^{\mathrm{T}}\boldsymbol{X}_0\boldsymbol{X}_0^{\mathrm{T}}\boldsymbol{v} = \lambda\boldsymbol{X}_0^{\mathrm{T}}\boldsymbol{v} \ . \tag{5.33}$$

This shows that if $\boldsymbol{v}$ is an eigenvector of the $\ell \times \ell$ matrix $\boldsymbol{X}_0 \boldsymbol{X}_0^{\mathrm{T}}$ with eigenvalue λ, then $\boldsymbol{X}_0^{\mathrm{T}} \boldsymbol{v}$ is a (not normalized) eigenvector of $\boldsymbol{S} = \boldsymbol{X}_0^{\mathrm{T}} \boldsymbol{X}_0$ with the same eigenvalue.

Thus, if $\ell < n$ it is possible to consider the smaller eigenvalue problem of decomposing $\boldsymbol{X}_0 \boldsymbol{X}_0^{\mathrm{T}}$. The equation $\boldsymbol{u}_i = \boldsymbol{X}_0^{\mathrm{T}} \boldsymbol{v}_i$ relates the ith largest eigenvalue of the smaller matrix to the ith principal component of $\boldsymbol{S}$.

5.4 Eigenspectrum and explained variance

Inspecting the eigenvalues of the empirical covariance matrix of sample data provides insights about the structure of the data, in particular its "true" dimensionality.

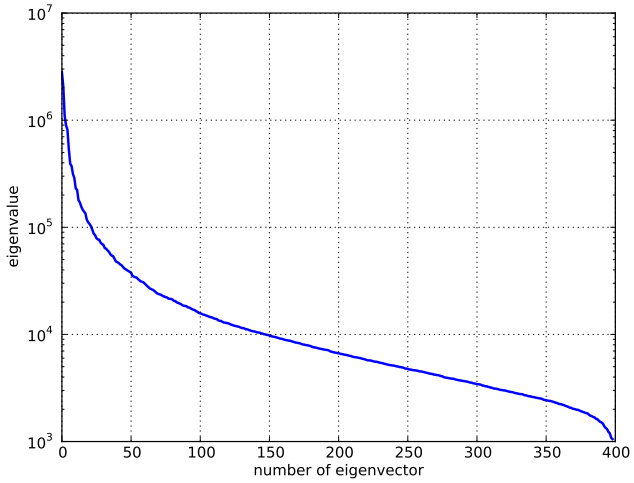

Fig. 5.8. Eigenvalues of the first 400 principal components of the Cambridge Face Database.

The trace of a square matrix is the sum of its eigenvalues. So the variability of the data can be expressed as $\mathrm{tr}\{\boldsymbol{S}\} = \sum_{j=1}^{n} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j = \sum_{i=1}^{n} \lambda_i$. Thus, the variability of the projected data can be written as $\sum_{j=1}^{m} \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{S} \boldsymbol{u}_j = \sum_{i=1}^{m} \lambda_i$. The quotient

$$\frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \tag{5.34}$$

is called the "explained variance". It reflects how much of the variability of the data is captured by the lower-dimensional model.

In Figure 5.8 plots the eigenvalues of the faces (Example 5.2) in decreasing order. This is often referred to as the eigenspectrum of the data. It reveals how much variance ist explained by the first k components. This is importnat for choosing m for a PCA model in practice.

5.5 Exercises

Exercise 5.1. Show that the mean of the projected data points $\tilde{S}$ is equal to the projection of the mean of the data points S.

Exercise 5.2. Show that for a matrix $\boldsymbol{U}$ composed of orthonormal vectors we have $\boldsymbol{U}^{-1} = \boldsymbol{U}^{\mathrm{T}}$.

Exercise 5.3. Prove that the data covariance matrix has only non-negative eigenvalues.

Exercise 5.4. Prove that the data covariance matrix has a rank of at most ℓ. Start by considering a vector $\boldsymbol{x} \in \mathbb{R}^n$ and verify that $\boldsymbol{x}\boldsymbol{x}^T$ has rank one.

6

Outlook

In 2006, organizers and attendees of the IEEE International Conference on Data Mining (ICDM) voted for the top 10 data mining algorithms, those methods that they regarded as most influencial in thei reseach area [Wu et al., 2008, Wu and Kumar, 2009]. These were:

C4.5 decision trees: This algorithm learns classifiers represented as decision trees [Quinlan, 1993], where leaves represent classification outcomes and branches conjunctions of features that lead to those outcomes.

k-Means Clustering: This algorithm is described in Section 4.

Support vector machines (SVMs): Theses predictive models first map the input data to a feature space and then perform linear classifiction in this space trying to maximize the minimum distance from training patterns to the decision boundary [e.g., Cortes and Vapnik, 1995].

Apriori algorithm: This method efficiently finds frequent itemsets (subsets of features) in databases [Agrawal and Srikant, 1994].

Expectation–Maximization (EM) algorithm: The EM algorithm can be used to estimate of parameters in statistical models, where the models depend on unobserved (i.e., latent, hidden) variables [Dempster et al., 1977].

PageRank: The PageRank algorithm sorts webpages according to their relevance by counting in- and outgoing hyperlinks [Brin and Page, 1998, see also section 23.2 in the textbook by Garcia-Molina et al., 2008].

AdaBoost: The AdaBoost algorithm is a method for citeratively training and combinig classifiers [Freund and Schapire, 1995].

Nearest Neighbor classification: This algorithm is described in Section 2.

Naive Bayes: The naive Bayes classifier is a simple, yet often successful method assuming that given a particular class the individual features are independent of each other.

CART: The acronym stands for Classification and Regression Trees, decision trees with binary braches [Breiman, 1984].

A

Some mathematical background

A.1 Probability theory and statistics

An introduction to statistics and probability theory is far beyond the scope of this course. However, "Knowledge discovery from data is fundamentally a statistical endeavor" [Fayyad et al., 1996]. In the following, I will thus give an informal hand-waving introduction to the basic concepts used in these lecture notes. The first chapters of the textbook by Mitzenmacher and Upfal [2017], which as been used for several formulations in the following subsection, give a nice introduction to the basic concepts of probability.

A.1.1 Basic definitions

Probabilities are used to describe randomness and uncertainty, for example the outcome of tossing a coin or rolling a dice. To formalize statements about probabilistic processes, we need the concept of a *probability space*, which is also called *probability triple*, because it consists of three components.

The first component is the *sample space* Ω, which is the set of all possible outcomes (also called elementary events). The possible outcomes of tossing a coin may be $\Omega = \{\text{head}, \text{tail}\}$ and of rolling a standard dice $\Omega = \{1, 2, \ldots, 6\}$.

Next we need to define the corresponding allowable *events*. Events occur with a certain probability. An event is a subset of Ω. For example, the event that rolling our dice has the outcome 6 simply corresponds to $\{6\}$, the event that the dice shows an odd number to $\{1, 3, 5\}$, and the event that we roll a number larger than 3 to $\{4, 5, 6\}$. The set of all events (a set of sets) is typically denoted by $\mathcal{F}$.

Finally, the probability function p assigns a probability between 0 and 1 to each event. The value of $p(E)$ reflects how likely the event $E \in \mathcal{F}$ is or, in other words, how likely the event is true. If $p(E) = 0$, then the event E will never happen. If $p(E) = 1$, then the E will surely happen. Let us assume that Ω is finite for a moment. Let us further assume that we repeat a

random experiment – tossing the coin or rolling the dice – n times. Thus, we observe n outcomes. Let n_E denote the number of times event E occurred. Clearly, $\sum_{x\in\Omega} n_{\{x\}} = n$. Now let $n \to \infty$. Then the fraction n_E/n defines the probability of the event E, that is, $p(E) = \lim_{n\to\infty} n_E/n$. Clearly, we have $0 \leq p(E) \leq 1$ for each event E. It holds $p(\Omega) = 1$, because one of the elementary events must happen. In our dice example, we have $p(\{6\}) = \frac{1}{6}$ and $p(\{1,3,5\}) = p(\{4,5,6\}) = \frac{1}{2}$. For a fair coin, we have $p(\{\text{head}\}) = p(\{\text{tail}\}) = \frac{1}{2}$.

Let us summarize in a formal definition. A sample space consists of

1. a sample space Ω, which is the set of all possible outcomes (also called elementary events);
2. a set of events $\mathcal{F}$, where each event is a set containing zero or more outcomes;
3. a function, here called p, assigning probabilities to events, with the following properties:
   a) $\forall E \in \mathcal{F} : 0 \leq p(E) \leq 1$;
   b) $p(\Omega) = 1$;
   c) for any countable sequence $E_1, E_2, \ldots$ of pairwise mutually disjoint events (i.e., $E_i \cap E_j = \emptyset$ for all $i \neq j$) it holds $p(\cup_{i=1,2,\ldots} E_i) = \sum_{i=1,2,\ldots} P(E_i)$.

Next, we need to define what a *random variable* is. A random variable on a sample space Ω is a real-valued (measurable) function on Ω. A *discrete random variable* takes only a countable number of values. For example, we can have a random variable X on $\Omega = \{\text{head}, \text{tail}\}$ assigning the value 1 to head anf 0 to tail. For rolling a dice we can define a random variable taking values in $\{1, 2, \ldots, 6\}$ directly mapping the elementary outcome to the corresponding value.

We write $p(X = x)$ for the probability that the random variable X takes the value x:

$$p(X = x) = \sum_{z\in\Omega : X(z)=x} p(z)$$

Again, the value of $p(X = x)$ reflects how likely X has the value x or, in other words, how likely the event $X = x$ is true. If the random variable is clear from the context, we use the shorthand notation $p(x)$ for $p(X = x)$. To keep the notation uncluttered, the random variable can also be indicated by a subscript (i.e., $p_X(x) = p(X = x)$).

The *expectation* (the average outcome) of a discrete random variable X is given by

$$\mathbb{E}[X] = \sum_x xp(X = x) \ , \tag{A.1}$$

where the sum runs over the all values in the range of X. Its *variance*

$$\sigma_X^2 = \sum_x p(x)(x - \mathbb{E}[X])^2 = \mathbb{E}[(X - \mathbb{E}[X])^2 \tag{A.2}$$

measures the amount of variation within the values of the random variable. The square root of the variance σ_X^2 is called the *standard deviation* σ_X.

Conditional, joint, and independent distributions

Now consider random variables X_1 and X_2 corresponding to two dices, each taking values in $\{1, 2, \ldots, 6\}$. We define a random variable Z corresponding to the sum of the two dices rolled independently. It takes values in $\{2, 3, \ldots, 12\}$. Now, what is the probability $p(Z = 4)$ that the sum is 4? There are 36 possible outcomes, $\{(1,1), (1,2), \ldots, (1,6), (2,1), (2,2), \ldots (6,6)\}$. The 3 outcomes $\{(1,3), (2,2), (3,1)\}$ lead to a sum of 4, so $p(Z = 4) = \frac{3}{36} = \frac{1}{12}$.

Now we consider the case that we already know the value x_1 of the first dice X_1. Then we write $p(Z = z \,|\, X_1 = x_1)$ for the probability that Z takes the value z given that X_1 has the value x_1. This expression is called a *conditional* probability, because it denotes the probability of $Z = z$ under the condition that $X_1 = x_1$. In our dices example, $p(Z = 2) = \frac{1}{36}$, $p(Z = 2 \,|\, X_1 = 1) = \frac{1}{6}$, and $p(Z = 2 \,|\, X_1 = 3) = 0$.

The *joint* probability $p(X_1 = x_1, X_2 = x_2)$ is the probability that both $X_1 = x_1$ and $X_2 = x_2$ are true. It holds $p(X_1 = x_1, X_2 = x_2) = p(X_1 = x_1 \,|\, X_2 = x_2)p(X_2 = x_2) = p(X_2 = x_2 \,|\, X_1 = x_1)p(X_1 = x_1)$. The two random variables are called *independent* if and only if $p(X_1 = x_1, X_2 = x_2) = p(X_1 = x_1)p(X_2 = x_2)$. This means that the sampling of X_1 does not depend on the sampling of X_2 and vive versa. If the variables are independent, it holds $p(X_1 = x_1 \,|\, X_2 = x_2) = p(X_1 = x_1)$ as well as $p(X_2 = x_2 \,|\, X_1 = x_1) = p(X_2 = x_2)$. If X_1 and X_2 are the values of dice rolls and Z denotes their sum, X_1 and X_2 are independent, but Z and X_1 as well as Z and X_2 are not.

Continuous random variables

The previous examples had the property that the random variables could take only finite many values, 2 and 6 in case of the coin or the dice, respectively. Now consider a variable that can take any real value in the interval $[0, 1[$ without preferring one value over another, that is, the variable can take infinite many values. Now we have a problem. If each of these values gets a discrete probability larger than zero, then the probabilities will inevitably sum up to a value larger than 1, which does not make sense.

Obviously, we have to extend our concept. So, we cannot assign non-zero probabilities to the event that we draw a particular value from $[0, 1[$. This is intuitive, because clearly we can wait for ages to sample by change exactly, say, $\pi/4$. However, because our random variable does not prefer any event in $[0, 1[$ over another, we can state that the probability of sampling a value in $[0, \frac{1}{2}[$ is 0.5.

We describe *continuous random variables* defined on a continuous sample space Ω by their *densities*. For simplicity, let us assume that $\Omega = \mathbb{R}$. A density

is a real-valued function p giving the probability that the value of a random variable x falls into a particular interval $[a, b]$ by

$$p(x \in [a, b]) = \int_a^b p(x)\mathrm{d}x \ . \tag{A.3}$$

Note that we use the same notation for a probability (left-hand side) and a density (right-hand side). This is ugly, because these are different mathematical objects. However, this extremely sloppy notation allows to state and derive results which hold for both discrete and continuous random variables using a single notation.

We have $\int_\Omega p(x)\mathrm{d}x = 1$ and, because negative probabilities make no sense, $p(x) \geq 0$. For random vectors $\boldsymbol{x}$ taking values in $\Omega = \mathbb{R}^n$, $n > 1$, we extend this concepts to multivariate densities with $\int_\Omega p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$ and $p(\boldsymbol{x}) \geq 0$.

The mean of continuous random variable $\boldsymbol{x}$ is given by

$$\mathbb{E}[\boldsymbol{x}] = \int_\Omega \boldsymbol{x} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \ . \tag{A.4}$$

The variance of a continuous random variable x taking values in $\mathbb{R}$ is accordingly

$$\sigma_x^2 = \int_\Omega p(x)(x - \mathbb{E}[\boldsymbol{x}])^2\mathrm{d}x = \mathbb{E}[(X - \mathbb{E}[x])^2] \ . \tag{A.5}$$

For random vectors we will extend the concept of variance to covariance in the next section.

A.1.2 Gaussian Distribution

One of the most important continuous distributions is the *Gaussian* or *normal* distribution. Its density function is given by

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ . \tag{A.6}$$

The Gaussian distribution has two parameters μ and σ, which correspond to the mean and the standard deviation, respectively. If $\mu = 0$ and $\sigma = 1$ it is called the standard normal distribution. The Gaussian density function is plotted for three different values of σ in Figure A.1.

The normal distribution plays an important role in modeling for several reasons. Most importantly, the Gaussian distribution arises in the *central limit theorem*. This theorem states that the probability distribution of a sum of n i.i.d. random variables with finite mean and variance approaches a Gaussian distribution with increasing n. The abbreviation i.i.d. stands for identically independently distributed meaning that the distributions of the n random variables are the same but that they are sampled independently of each other. Thus, if some outcome depends on several sources of randomness (and we

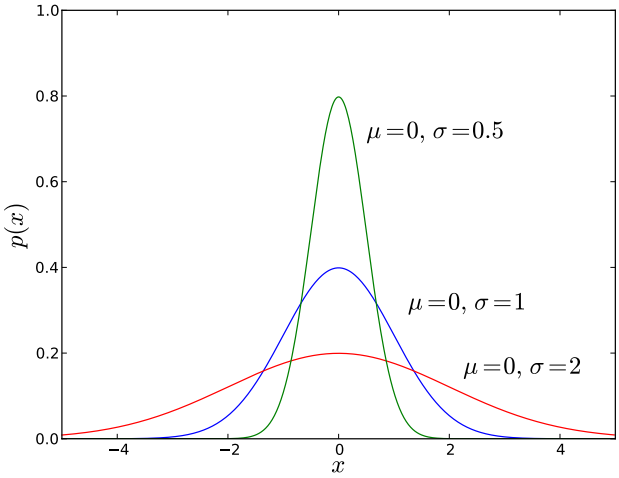

Fig. A.1. Normal distributions with zero mean and three different standard deviations.

assume that these sources add up) it may be well described by a Gaussian. Further, among all distributions having some given mean and variance, the Gaussian distribution has the highest entropy. This means, if we on the one hand can or want to fix mean and variance of a probabilistic model and on the other hand want to express maximum uncertainty about the outcomes, then we again arrive at the Gaussian distribution.

A.1.3 Covariance

The covariance matrix $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ of a random vector $\boldsymbol{x} \in \mathbb{R}^n$ is given by

$$\boldsymbol{C} = \mathbb{E}\left[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^{\mathrm{T}}\right] \quad . \tag{A.7}$$

Note that the right hand side is the expectation of an outer product. The diagonal entries c_{ii} correspond to the variances of the components x_i. The variance of x_i is a measure of how strongly x_i varies around it expectation. The off-diagonal entries measure the covariances, where $c_{ij} = c_{ji}$ indicates how strongly x_i and x_j vary together. If x_i and x_j are independent, we have $c_{ij} = 0$ (but not vice versa). If c_{ij} is positive, x_i and x_j are correlated, if c_{ij} is negative, the two variables are anti-correlated (i.e., if x_i is large, x_j tends to be low and vice versa).

A.2 Derivatives

You need basic calculus for understanding data mining algorithms. This text cannot provide an introduction to calculus, so you have to go back to your notes from school or grap one of the numerous textbooks that are around. This section can, however, be used to brush up on using derivatives.

A.2.1 Definition via difference quotient

Let $f : \mathbb{R} \to \mathbb{R}$ be a real-valued function. If the limit

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{h} \tag{A.8}$$

exists, the function f is differentiable and f' is its derivative. If you consider the graph of the function f, then $f'(x)$ is the slope of the graph at point x. Informally, $f'(x)$ describes how $f(x)$ changes if you increase its argument x a tiny little bit. If the amount of $f'(x)$ is large, then the change is large. If the amount is small, then the change is small. If the $f'(x)$ is positive, then the change will increase $f(x)$. If it is negative, it will lead to a decrease. If $f(x)$ is a local maximum or minimum, $f'(x) = 0$ (not necessarily the other way round, as $f'(x) = 0$ could also indicate a saddle point).

A.2.2 Differentiation rules

Now we review basic rules for computing derivatives. For the remainder of this section A.2.2, let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be differentiable functions.

Sum rule. If $h(x) = f(x) + g(x)$ then h is differentiable and $h'(x) = f'(x) + g'(x)$.

Chain rule. If $h(x) = f(g(x))$ then h is differentiable and $h'(x) = f'(g(x))g'(x)$.

Constant factor rule. If $h(x) = af(x)$ with $a \in \mathbb{R}$, then h is differentiable and $h'(x) = af'(x)$. This follows from the chain rule and the fact the derivative of a constant function $g(x) = a$ is zero.

Product rule. If $h(x) = f(x)g(x)$ then h is differentiable and $h'(x) = f'(x)g(x) + f(x)g'(x)$.

Power rule. The product rule gives us for $f(x) = x^n$ with integer n the derivative $f'(x) = nx^{n-1}$.

Reciprocal rule. If

$$h(x) = \frac{1}{f(x)} \tag{A.9}$$

for f non-vanishing (i.e., f is never zero) then h is differentiable and

$$h'(x) = -\frac{f'(x)}{[f(x)]^2} \ . \tag{A.10}$$

This implies $g'(x) = -1/x^2$ for $g(x) = 1/x$.

Quotient rule. Putting chain rule and reciprocal rule together, we get for

$$h(x) = \frac{f(x)}{g(x)} \tag{A.11}$$

and non-vanishing g the derivative

$$h'(x) = -\frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2} \ . \tag{A.12}$$

Derivative of exponential and logarithmic function. For $f(x) = e^x$ we have $f'(x) = e^x$ and for $f(x) = \ln x$ we have $f'(x) = 1/x$.

A.2.3 Partial derivatives

To deal with functions that depend on several variables, we introduce the concept of a partial derivative. The partial derivative of a function f with respect to a variable x is denoted by

$$\frac{\partial}{\partial x} f = \frac{\partial f}{\partial x} \tag{A.13}$$

and is defined as the derivative of f with respect to x with all other variables f depends on held constant.

In the case that f just depends on a single variable x, we simply have

$$\frac{\partial}{\partial x} f(x) = f'(x) \ . \tag{A.14}$$

Now let $f : \mathbb{R}^n \to \mathbb{R}$ depend on n variables $(x_1, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$. Then the partial derivative of f with respect to x_i at a point $\boldsymbol{x}_0 \in \mathbb{R}^n$ is defined as

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x}_0) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{e}_i) - f(\boldsymbol{x})}{\epsilon} \tag{A.15}$$

with $\boldsymbol{e}_i$ being the ith unit vector (i.e., a vector where the ith component is one and all other components are zero) given that the limit exists. We collect these partial derivatives in the *gradient*. The gradient $\nabla f(\boldsymbol{x})$ at $\boldsymbol{x} \in \mathbb{R}^n$ is defined as

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \partial f(\boldsymbol{x})/\partial x_1 \\ \partial f(\boldsymbol{x})/\partial x_2 \\ \vdots \\ \partial f(\boldsymbol{x})/\partial x_n \end{pmatrix} \ . \tag{A.16}$$

The gradient is related to the *rate of change* of the function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x} \in \mathbb{R}^n$ when moving in the direction $\boldsymbol{u} \in \mathbb{R}^n$, $\|\boldsymbol{u}\| = 1$. The rate of change is defined as:

$$\nabla_{\boldsymbol{u}} f(\boldsymbol{x}) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{u}) - f(\boldsymbol{x})}{\epsilon}$$

Then the gradient $\nabla f(\boldsymbol{x})$ points in the direction $\nabla f(\boldsymbol{x})/\|\nabla f(\boldsymbol{x})\|$ giving maximum rate $\|\nabla f(\boldsymbol{x})\|$ of change. This generalizes the relation the derivative of a real-valued function depending only on a single variable to the slope of its graph.

Example A.1. Let $f(x, y) = x^2 - 2xy + y^2$. Then $\frac{\partial}{\partial x} f(x, y) = 2x - 2y$ and $\frac{\partial}{\partial y} f(x, y) = 2y - 2x$. That is, we have

$$\nabla f(x, y) = \begin{pmatrix} 2x - 2y \\ 2y - 2x \end{pmatrix} \quad . \tag{A.17}$$

A.3 Treating equality constraints

Suppose we want to minimize some function $f : \mathbb{R}^n \to \mathbb{R}$ subject to the constraint $c(\boldsymbol{x}) = 0$. This means, we want to optimize f, but the solution $\boldsymbol{x}^\star$ must have the property $c(\boldsymbol{x}^\star) = 0$.

The general framework of constraint optimization – in particular the fundamental method of Lagrange multipliers – is beyond the scope of this introductory text. I refer to advanced lectures on optimization, my lecture notes for "Machine Learning: Kernel-based Methods", and to the many textbooks on optimization (e.g., the book by Lange [2004]). However, the simple type of problems with equality constraints we are considering here is straight-forward.

As an example, let us consider the objective function $f(\boldsymbol{x}) = x_1^2 + x_2^2$ subject to the constraint $c(\boldsymbol{x}) = 1 + \frac{1}{2} x_1 - x_2 = 0$, which is illustrated in Figure A.2. The points on the blue line are feasible points, that is, elements of the search space that fulfill the constraint. Both the green (more to the left) as well as the red (more to the right) point are feasible. However, the green point is not optimal. The gradient $\nabla f(\boldsymbol{x}) = (\partial f(\boldsymbol{x})/\partial x_1, \partial f(\boldsymbol{x})/\partial x_2)^{\mathrm{T}}$ (see section A.2.3) has a component that can be reduced by moving along the blue line (here by moving to the right). Only when the gradient of the objective function and the gradient of the constraint $\nabla c(\boldsymbol{x}) = (\partial c(\boldsymbol{x})/\partial x_1, \partial c(\boldsymbol{x})/\partial x_2)^{\mathrm{T}}$ are parallel, every small move along the constraint surface (the blue line) will worsen the solution. Thus, a *necessary* condition at an optimal point $\boldsymbol{x}^\star$ is that the two gradients are parallel. This is the case when

$$\nabla f(\boldsymbol{x}^\star) = \lambda \nabla c(\boldsymbol{x}^\star) \tag{A.18}$$

for some scalar λ. At the red point in Figure A.2 this condition is fulfilled and therefore the point is a local optimum. The real-valued variable λ is called a *Lagrange multiplier*. If there are several constraints, we get an equation such as (A.18) with an individual Lagrange multiplier for each constraint.

Now we can compute the constrained minimum of our example problem. The condition $\nabla f(\boldsymbol{x}) = \lambda \nabla c(\boldsymbol{x})$ gives $(2x_1, 2x_2)^{\mathrm{T}} = \lambda \left(\frac{1}{2}, -1\right)^{\mathrm{T}}$ implying

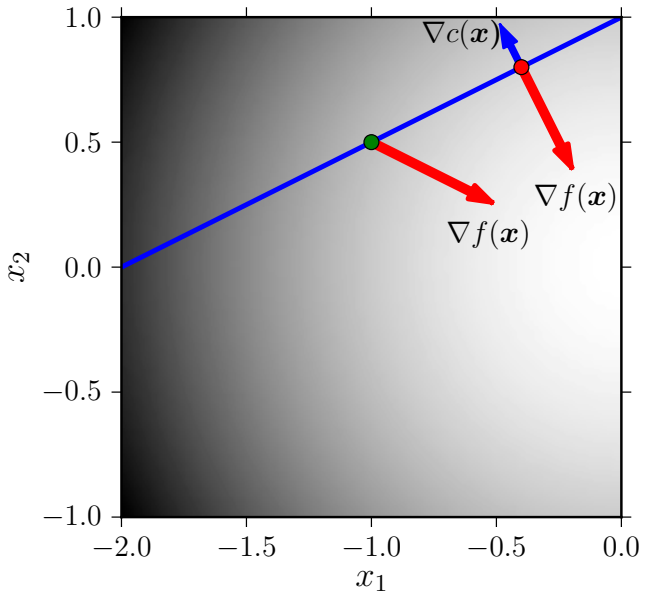

Fig. A.2. Two-dimensional constrained optimization problem. We want to minimize the function (i.e., find the x_1 and x_2 with the lightest color) but are constrained to the points on the blue line.

$x_2 = -2x_1$. Plugging this into the equality constraint gives $x_1^\star = -\frac{2}{5}$ and $x_2^\star = \frac{4}{5}$.

Now, what happens if we have multiple equality constraints c_i, $i = 1, \ldots, k$? At an optimal point $\boldsymbol{x}^\star$, the gradient $\nabla f(\boldsymbol{x}\star)$ should have no component which is perpendicular to all $\nabla c_i(\boldsymbol{x}\star)$. Otherwise, we could move a tiny little bit in this direction without violating the constraints. Thus, $\nabla f(\boldsymbol{x}\star)$ must be a linear combination – lie in she span – of the $\nabla c_i(\boldsymbol{x}\star)$, that is, there are Lagrange multiplier $\lambda_1, \ldots, \lambda_k$ such that

$$\nabla f(\boldsymbol{x}^\star) = \sum_{i=1}^{k} \lambda_i \nabla c_i(\boldsymbol{x}\star) \ . \tag{A.19}$$

This is just a necessary condition. Clearly, it must be ensured that $\boldsymbol{x}^\star$ fulfills the conditions (i.e., $c_i(\boldsymbol{x}\star) = 0$, $i = 1, \ldots, k$).

References

R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, volume 1215, pages 487–499. Morgan Kaufmann, 1994.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

L. Breiman. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

A. P. Dempster, N. M. Laird, D. B. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2 edition, 2008.

J. Ghosh and A. Liu. *K-Means*, chapter 2. In Wu and Kumar [2009], 2009.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

K. Lange. *Optimization*. Springer Texts in Statistics. Springer-Verlag, 2004.

S. Lloyd. Last square quantization in PCM. *IEEE Transactions in Information Theory*, 28:129–137, 1982. Journal version of a *Bell Telephone Laboratories Paper* from 1957.

M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017.

J. Quinlan. *C4. 5: Programs for machine learning*. Morgan Kaufmann, 1993.

F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *IEEE Workshop on Applications of Computer Vision*, pages 138–142. IEEE Computer Society Press, 1994.

M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

X. Wu and V. Kumar, editors. *The top ten algorithms in data mining*. Chapman & Hall/CRC, 2009.

X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.